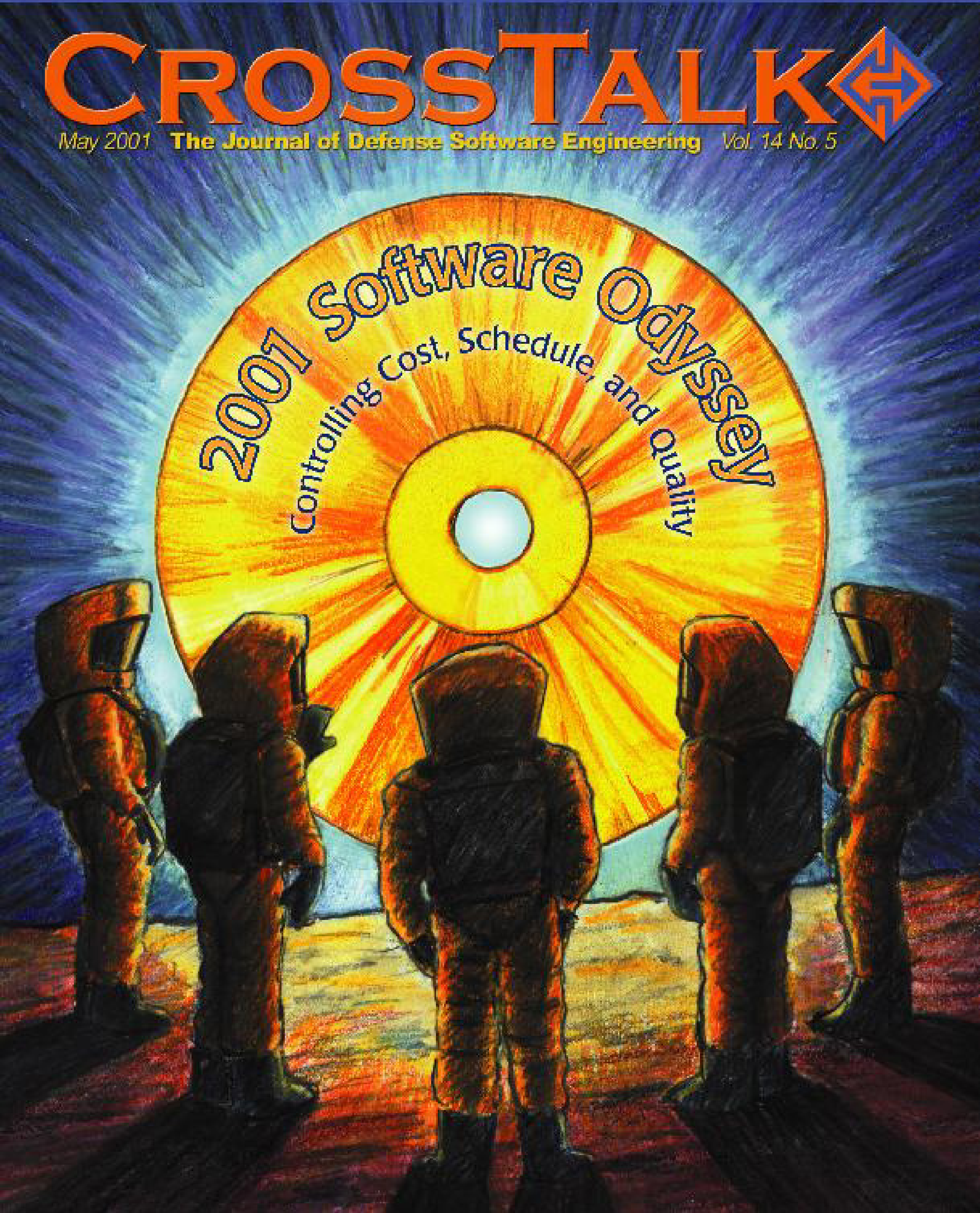
Special Software Technology Conference Issue

CrossTalk

May 2001 The Journal of Defense Software Engineering Vol. 14 No. 5



Report Documentation Page	*Form Approved* *OMB No. 0704-0188*

1. REPORT DATE **MAY 2001**	2. REPORT TYPE	3. DATES COVERED **00-00-2001 to 00-00-2001**

4. TITLE AND SUBTITLE **CrossTalk. The Journal of Defense Software Engineering. Volume 14, Number 5, May2001**	5a. CONTRACT NUMBER
	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)	5d. PROJECT NUMBER
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **517 SMXS MXDEA,6022 Fir Ave,Hill AFB,UT,84056-5820**	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

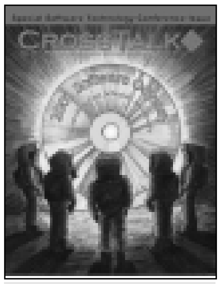
12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

13. SUPPLEMENTARY NOTES

14. ABSTRACT

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT **unclassified**	b. ABSTRACT **unclassified**	c. THIS PAGE **unclassified**	**Same as Report (SAR)**	**36**	



On the Cover: Barclay Tucker is a free-lance illustrator. His company is Designer Under The Stairs.

Departments

CrossTalk

SPONSOR	*Lt. Col. Glenn A. Palmer*
PUBLISHER	*Tracy Stauder*
ASSOCIATE PUBLISHER	*Elizabeth Starrett*
MANAGING EDITOR	*Pam Bowers*
ASSOCIATE EDITOR	*Benjamin Facer*
ARTICLE COORDINATOR	*Nicole Kentta*
CREATIVE SERVICES COORDINATOR	*Janna Jensen*
VOICE	801-586-0095
FAX	801-777-5633
E-MAIL	crosstalk.staff@hill.af.mil
CROSSTALK ONLINE	www.stsc.hill.af.mil/Crosstalk/crostalk.html
CRSIP ONLINE	www.crsip.hill.af.mil

Subscriptions: Send correspondence concerning subscriptions and changes of address to the following address. You may e-mail or use the form on p. 18.

Ogden ALC/TISE
7278 Fourth St.
Hill AFB, Utah 84056-5205

Article Submissions: We welcome articles of interest to the defense software community. Articles must be approved by the CROSSTALK editorial board prior to publication. Please follow the Author Guidelines, available at www.stsc.hill.af.mil/CrossTalk/xtlkguid.pdf. CROSSTALK does not pay for submissions. Articles published in CROSSTALK remain the property of the authors and may be submitted to other publications.

Reprints and Permissions: Requests for reprints must be requested from the author or the copyright holder. Please coordinate your request with CROSSTALK.

Trademarks and Endorsements: This DoD journal is an authorized publication for members of the Department of Defense. Contents of CROSSTALK are not necessarily the official views of, or endorsed by, the government, the Department of Defense, or the Software Technology Support Center. All product names referenced in this issue are trademarks of their companies.

Coming Events: We often list conferences, seminars, symposiums, etc. that are of interest to our readers. There is no fee for this service, but we must receive the information at least 90 days before registration. Send an announcement to the CROSSTALK Editorial Department.

STSC Online Services: at www.stsc.hill.af.mil

Call 801-777-7026, e-mail: randy.schreifels@hill.af.mil

Back Issues Available: The STSC sometimes has extra copies of back issues of CROSSTALK available free of charge.

The Software Technology Support Center was established at Ogden Air Logistics Center (AFMC) by Headquarters U.S. Air Force to help Air Force software organizations identify, evaluate, and adopt technologies to improve the quality of their software products, efficiency in producing them, and their ability to accurately predict the cost and schedule of their delivery.

Start Your Software Odyssey Here

If you've been on an odyssey (extended journey) looking for the magic formula to successful software intensive systems acquisition — stop, relax, and read this edition of CrossTalk. After 15 plus years of formal software process improvement, numerous case studies, and more review boards and task forces than we care to list … the secret is now revealed! Get a good start, do quality work, stay on the right course, and work as a team. So simple? Not quite — Each step requires a body of knowledge, talented people, superb leadership, and adequate funding to fuel the program engine.

A good start requires disciplined requirements generation and plotting the right course through sound systems engineering, coupled with the appropriate acquisition strategy. In their article, *The Spiral Model as a Tool for Evolutionary Acquisition*, Dr. Barry Boehm and Wilfred J. Hansen define the relationship between evolutionary acquisition strategies and spiral development. They also describe the six essential aspects that every spiral process must exhibit and some "hazardous look-alikes" that must be recognized. This is a must-read article that summarizes a large body of knowledge related to successful program planning and management.

Obtaining talented people is one of the top issues for most large software development organizations. Retention of software engineers continues to be a major problem for the U.S. Air Force Air Logistics Centers. Contractors in major technology centers find retention even more difficult. In the article *The Next Refinement to Software*, Keith Thurston of the General Services Administration, Office of Government-Wide Policy, outlines some actions required to ensure technology is accessible to individuals with disabilities. Designing systems for accessibility is not only an issue of legal compliance, it's also a means to better employ the talents of the 54 million Americans with disabilities.

Because large software-intensive system developments require a myriad of organizations with different interests (e.g., contractors, users, maintainers, technologists) to function as a team, above par leadership is essential. In their article, *Managing the Invisible*, RADM Patrick Moneymaker and Dr. Lynn Robert Carter outline five principles of leading high-performance teams. They go further by relaying their real-world illustrations of the principles, showing congruence with the best of management thought, and then relate them to software process improvement.

Cost reduction programs have become essential to an organization's ability to maintain sufficient resources to serve customers and accomplish their mission. Steve Perkins, in his article *Streamlined Networking Brings Oracle Big Savings, Better Service,* provides insight to how technology giant, Oracle, changed the way it managed information to reduce infrastructure cost. While exploiting the capabilities of technology to reduce operating costs has been a constant through the ages, the pace of change and number of people involved has certainly increased. The article describes Oracle's approach to centralizing system control while concurrently providing employees improved access to information.

Driven by increased computing power and the desire for more functions, software efforts grow in size and complexity, which in turn increases the risk of software induced problems. NASA's Dr. Linda H. Rosenberg describes how the agency is approaching this issue in *Verification and Validation Implementation at NASA.* Establishing a center of expertise, defining criteria for applying IV&V, and then identifying the projects to participate based on risk are the steps taken by NASA to improve. Likewise, this trend toward more complexity demands more efficient acceptance testing. Automating the validation test (e.g., input control, data collection, and recording) is underway at many facilities. Prototype systems developed by the Air Force Research Laboratory have demonstrated how weeks of testing can be reduced to hours. In their article, *Maintaining the Quality of Black-Box Testing*, Patrick J. Schroeder and Dr. Bogdan Korel of the Illinois Institute of Technology, illustrate how combinatorial test designs reduce test efforts without sacrificing quality.

We hope this issue will help bring your cost, schedule, and quality odyssey to a successful conclusion.

Glenn A. Palmer

Lt. Col. Glenn A. Palmer
Director, Computer Resources Support Improvement Program

The Spiral Model as a Tool for Evolutionary Acquisition

Dr. Barry Boehm
University of Southern California
Center for Software Engineering

Wilfred J. Hansen
Software Engineering Institute

Recent Department of Defence policy embodied in the new 5000.1 and 5000.2 series of acquisition regulations strongly emphasizes evolutionary acquisition and the use of spiral development for software. Spiral development has been used successfully in many commercial systems and in a good number of defense systems. But some ambiguities in previous spiral model definitions has also led to a good number of unsuccessful projects adopting "hazardous spiral look-alikes." This paper provides clearer definitions of a set of six Spiral Model Essentials or critical success factors for spiral development. It illustrates each with examples, identifies the hazardous look-alikes to avoid, and provides guidelines for using the spiral model in support of evolutionary acquisition.

Since its original publication [1], the spiral development model diagrammed in Figure 1 has been used successfully in many defense and commercial projects. To extend this base of success, the Department of Defense (DoD) has recently rewritten the defense acquisition regulations to incorporate "evolutionary acquisition," an acquisition strategy designed to mesh well with spiral development. In particular, DoD Instruction 5000.2 subdivides acquisition [2]:

> "There are two ... approaches, evolutionary and single step to full capability. An evolutionary approach is preferred. … [In this] approach, the ultimate capability delivered to the user is divided into two or more blocks, with increasing increments of capability." (p. 20)

Here, a block corresponds to a single product release. The text goes on to specify the use of spiral development within blocks:

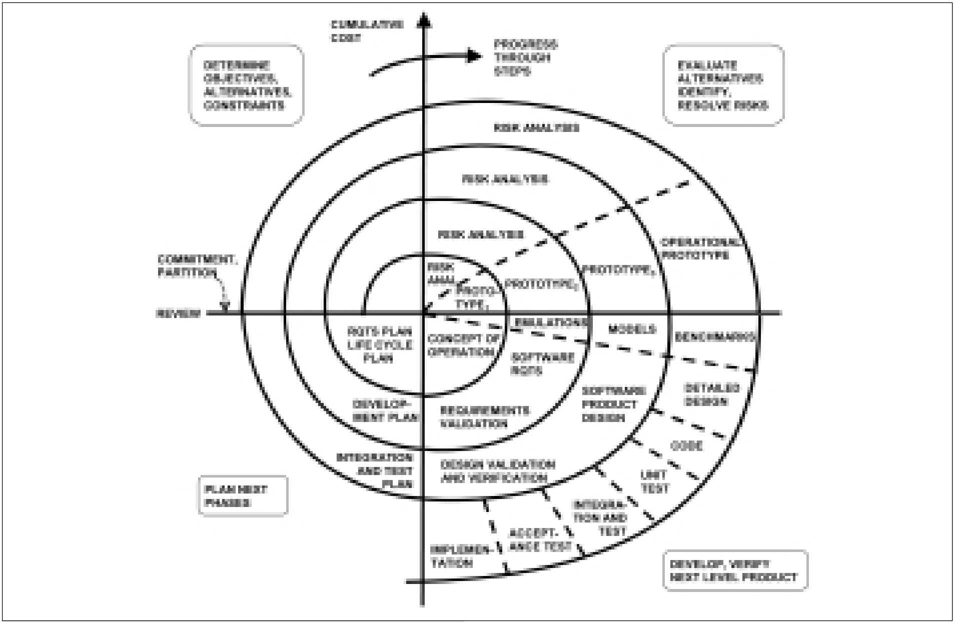
> "For both the evolutionary and single-step approaches, software development shall follow an iterative spiral development process in which continually expanding software versions are based on learning from earlier development." (p. 20)

Given this reliance on the spiral development model, an in-depth definition is appropriate. Two recent workshops provided one.

The University of Southern California (USC) Center for Software Engineering and the Carnegie Mellon University Software Engineering Institute held two workshops last year to study spiral development and identify a set of critical success factors and recommended approaches. Their results appear in two reports, [3, 4] and are available on the workshop Web site www.sei.emu.edu/cbs/spiral2000

The first author's presentations at these workshops defined spiral development and are followed below. The definition was first converted to a report [5], where details, suggestions, and further references can be found. Additionally, a follow-on article appearing in a later CROSSTALK issue, will address the relationships among spiral development, evolutionary acquisition, and the Integrated Capability Maturity Model.

Figure 1: *Original Spiral Development Diagram*



Spiral Development Definition and Context

We can begin with a high-level definition of the spiral development model:

> The spiral development model is a *risk-driven process model* generator that is used to guide multi-stakeholder concurrent engineering of software-intensive systems. It has two main distinguishing features. One is a *cyclic* approach for incrementally growing a system's degree of definition and implementation while decreasing its degree of risk. The other is a set of *anchor point milestones* for ensuring stakeholder commitment to feasible and mutually satisfactory system solutions.

The highlighted terms deserve further explanation:

- *Risks* are situations or possible events that can cause a project to fail to meet its goals. They range in impact from trivial to fatal and in likelihood from certain to improbable. Since risk considerations dictate the path a development must take, it is important that those risks be cataloged candidly and completely. See the references for taxonomy of risks [6] and a method for identifying them [7].
- A *process model* answers two main questions: What should be done next? How long should it continue? Under the spiral model the answers to these questions are driven by risk considerations and

vary from project to project and sometimes from one spiral cycle to the next. Each choice of answers generates a different process model.

- The *cyclic* nature of the spiral model is illustrated in Figure 1. Rather than develop the completed product in one step, multiple cycles are performed with each taking steps calculated to reduce the most significant remaining risks.
- Each *anchor point milestone* is a specific combination of artifacts and conditions that must be attained at some point. The sequence of three anchor point milestones — "LCO," "LCA," and "ICO" — is defined in Spiral Essential 5 (to be discussed). These milestones impel the project toward completion and offer a means to compare progress between one project and another.

Many aspects of spiral development are omitted in the above definition. The remainder of this paper expands the definition by describing six essential aspects that every proper spiral process must exhibit. The essentials are sketched in Figure 2. Each subsequent section describes a Spiral Essential, the critical success factor, reasons why it is necessary, and the variant process models it allows. Examples are given. Other process models that are precluded by the Spiral Essential are described. Because these may seem to be instances of the spiral model, but lack necessary essentials and thus risk failure, they are called "hazardous spiral look-alikes."

Spiral Essential 1: Concurrent Determination of Key Artifacts (Operational Concept, Requirements, Plans, Design, Code)

For a successful spiral effort, it is vital to determine balanced portions of certain key artifacts concurrently and not sequentially. These key artifacts are the operational concept, the system and software requirements, the plans, the system and software architecture and design, and the code components, including COTS, reused components, prototypes, success-critical components, and algorithms. Ignoring this Essential by sequentially determining the key artifacts will prematurely overconstrain the project, and often extinguish the possibility of developing a product satisfactory to the stakeholders.

Variants: Within the constraints of this Essential, variation is possible in the product and process internals of the concurrent engineering activity. For a low technology, interoperability-critical system, the initial spiral products will be requirements-intensive. For a high technology, more standalone system, the initial spiral products will be prototype-code intensive. Also, the Essential does not dictate the number of mini-cycles (e.g., individual prototypes for COTS, algorithms, or user-interface risks) within a given spiral cycle.

Example: One-Second Response Time

Examples of failure due to omission of this Essential include premature commitments to hardware platforms, incompatible combinations of COTS components, and requirements whose achievability has not been validated. For instance, in the early 1980s, a large government organization contracted with TRW to develop an ambitious information system for more than a thousand users. This system was to be distributed across a campus and offer powerful query and analysis access to a large and dynamic database. Based largely on user need surveys and an oversimplified high-level performance analysis, TRW and the customer fixed into the contract a requirement for a system response time of less than one second.

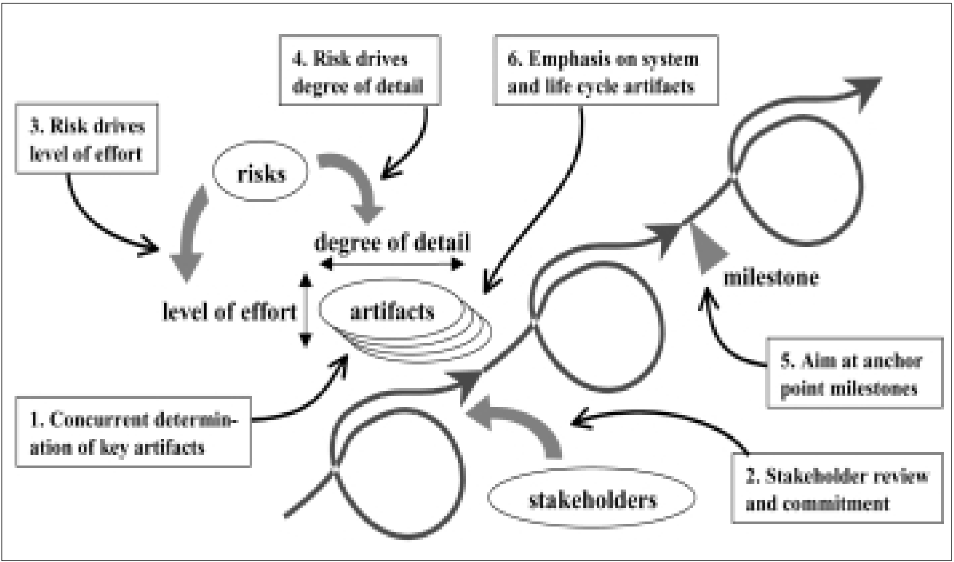
Two thousand pages of requirements later, the software architects found that subsecond performance could only be provided via a highly customized design that attempted to cache data and anticipate query patterns to be able to respond to each user within one second. The resulting hardware architecture had more than 25 super-minicomputers caching data according to algorithms whose actual performance defied easy analysis. The estimated cost was $100 million; see the upper arc in Figure 3 (See page 6).

Faced with this exorbitant cost, the customer and developer decided to develop and user-test a prototype. The results showed a four-second response time with 90 percent user satisfaction. This lower performance could be achieved with a modified client-server architecture, cutting development costs to $30 million as shown by the lower arc in the Figure 3 [8]. Concurrent prototyping would have saved two years' time and large-project effort.

Hazardous Spiral Look-Alike: Violation of Waterfall Assumptions

Essential 1 excludes the use of an incremental sequence of waterfall developments in the common case where there is a high risk of violating the assumptions underlying the waterfall model. These assumptions are that the requirements are *pre-specifiable*, *slowly changing*, and satisfactory to all stakeholders, and that a well understood *architecture* can meet these requirements. These assumptions must be met by a project if the waterfall model is to succeed. If all are true, then it is a project risk *not* to specify the requirements: the spiral-dictated risk analysis results in a waterfall approach for this project. If any assumption is false, then a waterfall approach will commit the project to troublesome assumptions and requirements

Figure 2: *Pictorial Sketch of the Six Spiral Essentials*


4. Risk drives degree of detail
6. Emphasis on system and life cycle artifacts
3. Risk drives level of effort
risks
degree of detail
milestone
level of effort
artifacts
5. Aim at anchor point milestones
1. Concurrent determination of key artifacts
2. Stakeholder review and commitment
stakeholders

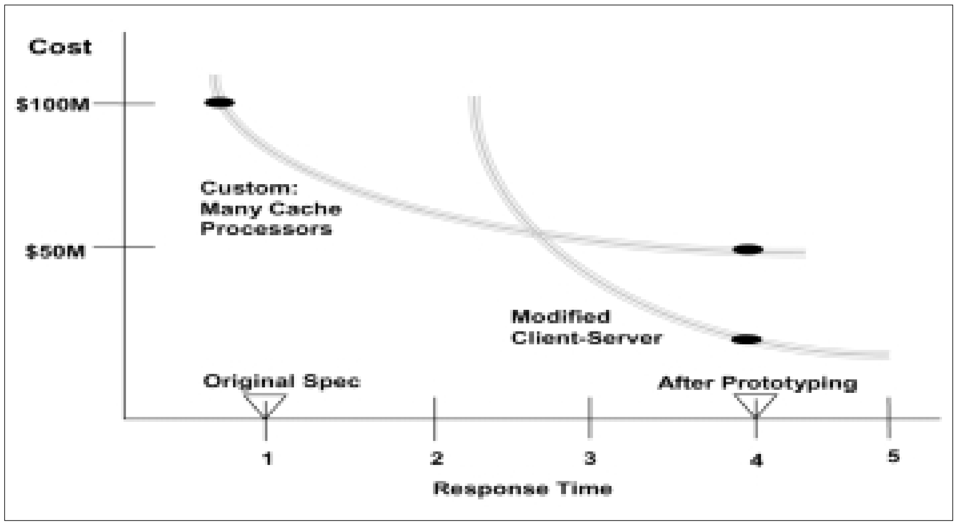


Figure 3: *Two System Designs: Cost vs. Response Time*

mismatches. Here are typical cases that violate waterfall assumptions:

- Requirements are not generally *pre-specifiable* for new user-interactive systems, because of the IKIWISI syndrome. When asked for their required screen layout for a new decision-support system, users will generally say, "I can't tell you, but I'll know it when I see it (IKIWISI)." In such cases, a concurrent prototyping/requirements/architecture approach is necessary.
- *Rapidly changing* requirements are well illustrated by electronic commerce projects, where the volatility of technology and the marketplace is high. The time it takes to write detailed requirements is not a good investment of the scarce time-to-market available when it is likely the requirements will change more than once downstream.
- The *architecture* and its implications were the downfall of the one-second response time example.

Spiral Essential 2: Each Cycle Does Objectives, Constraints, Alternatives, Risks, Review, and Commitment to Proceed

Spiral Essential 2 identifies the activities that need to be done in each spiral cycle. These include consideration of critical stakeholder objectives and constraints, elaboration and evaluation of project and process alternatives for achieving the objectives subject to the constraints, identification and resolution of risks attendant on choices of alternative solutions, and stakeholders' review and commitment to proceed based on satisfaction of their critical objectives and constraints. If all of these are not considered, the project may be prematurely committed to alternatives that are either unacceptable to key stakeholders or overly risky.

Variants: Spiral Essential 2 does not mandate particular generic choices of risk resolution techniques, although guidelines are available [9]. Nor does this Essential mandate particular levels of effort for the activities performed during each cycle. Levels must be balanced between the risks of learning too little and the risks of wasting time and effort gathering marginally useful information.

Example: Windows-Only COTS

Ignoring Essential 2 can lead to wasted effort in elaborating an alternative that could have been shown earlier to be unsatisfactory. One of the current USC digital library projects is developing a Web-based viewer for oversized artifacts (e.g., newspapers, large images). The initial prototype featured a tremendously powerful and high-speed viewing capability, based on a COTS product called ER Mapper. The initial project review approved selection of this COTS product, even though it only ran well on Windows platforms, and the Library had significant Macintosh and UNIX user communities. This decision was based on initial indications that Mac and UNIX versions of ER Mapper would be available "soon." These indications proved unreliable, however, and the anticipated delay became quite lengthy. So after wasting considerable effort on ER Mapper, it was dropped in favor of a less powerful but fully portable COTS product, Mr. SID. The excess effort could have been avoided had the review team included stakeholders from the Mac and UNIX communities on campus who would have done the necessary investigations earlier.

Hazardous Spiral Look-Alike: Excluding Key Stakeholders

Essential 2 excludes the process model of organizing the project into sequential phases or cycles in which key stakeholders are excluded. These omissions are likely to cause critical risks to go undetected. Examples are excluding developers from system definition, excluding users from system construction, or excluding system maintainers from either definition or construction. Excluding developer participation in early cycles can lead to project commitments based on unrealistic assumptions about developer capabilities. Excluding users or maintainers from development cycles can lead to win/lose situations, which generally devolve into lose-lose situations.

Spiral Essential 3: Level of Effort Driven by Risk Considerations

Spiral Essential 3 dictates the use of risk considerations to answer the difficult questions concerning how much is enough of a given activity such as domain engineering, prototyping, testing, configuration management, and so on. The recommended approach is to evaluate Risk Exposure (RE), which is computed as Probability (Loss) • Size (Loss). There is risk of project error RE_{error} from doing too little effort and project delay RE_{delay} from doing too much. Ideally, the effort expended will be that which minimizes the sum $RE_{error} + RE_{delay}$. This approach applies to most activities that are undertaken in a spiral development.

Variants: The variants to be considered include the choice of methods used to pursue activities (e.g., MBASE/WinWin, Rational RUP, JAD, QFD, ESP) and the degree of detail of artifacts produced in each cycle. Another variant is an organization's choice of particular methods for risk assessment and management.

Example: Pre-Ship Testing

Risk considerations can help determine "how much testing is enough" before shipping a product. The more testing that is

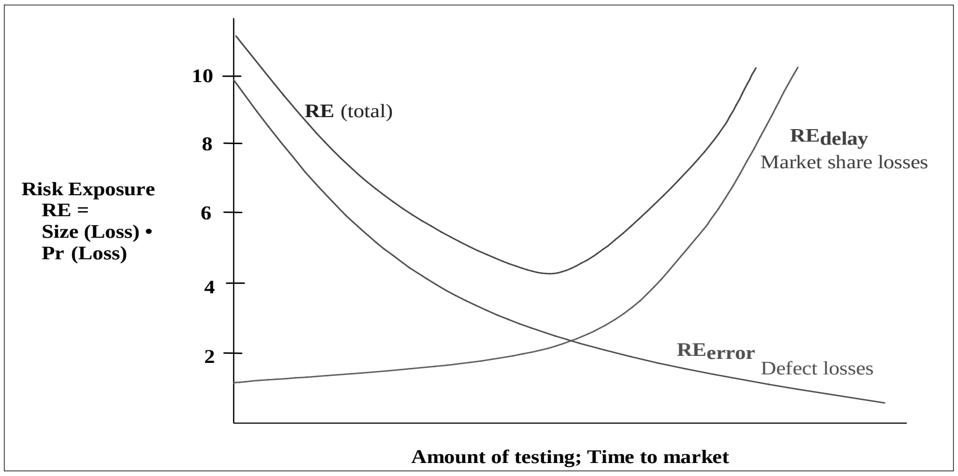


Figure 4: *Pre-Ship Test Risk Exposure*

done, the lower becomes RE_{error} due to defects, as discovered defects reduce both the size of loss due to defects and the probability that undiscovered defects still remain. However, the more time spent testing, the higher is RE_{delay} from losses due to both competitors entering the market and decreased profitability on the remaining market share. As shown in Figure 4, the sum of these risk exposures achieves a minimum at some intermediate level of testing. The location of this minimum-risk point in time will vary by type of organization. For example, it will be considerably shorter for a "dot.com" company than it will for a safety-critical product such as a nuclear power plant. Calculating the risk exposures also requires an organization to accumulate a fair amount of calibrated experience on the probabilities and size of losses as functions of test duration and delay in market entry.

Hazardous Spiral Look-Alikes: Risk Insensitivity

Hazardous spiral model look-alikes excluded by Essential 3 are:

- Risk-insensitive evolutionary development (e.g., neglecting scalability risks).
- Risk-insensitive incremental development (e.g., sub-optimizing during increment 1 with an ad hoc architecture that must be dropped or heavily reworked to accommodate future increments).
- Impeccable spiral plans with no commitment to managing the risks identified.

Spiral Essential 4: Degree of Detail Driven by Risk Considerations

Where Essential 3 circumscribes efforts, Essential 4 circumscribes the results of those efforts; it dictates that risk considerations determine the degree of detail of artifacts. This means, for example, that the traditional ideal of a complete, consistent, traceable, testable requirements specification is not a good idea for certain product components, such as a graphic user interface (GUI) or COTS interface. Here, the risk of precisely specifying screen layouts in advance of development involves a high probability of locking an awkward user interface into the development contract, while the risk of not specifying screen layouts is low, given the general availability of flexible GUI-builder tools. Even aiming for full consistency and testability can be risky, as it creates a pressure to prematurely specify decisions that would better be deferred (e.g., the form and content of exception reports). However, some risk patterns make it very important to have precise specifications, such as the risks of safety-critical interface mismatches between hardware and software components, or between a prime contractor's and a subcontractor's software.

This guideline shows when it is risky to over specify and under specify software features:

- If it's risky to *not* specify precisely, *DO specify* (e.g., hardware-software interface, prime-subcontractor interface).
- If it's risky to specify precisely, *DO NOT specify* (e.g., GUI layout, volatile COTS behavior).

Variants: Unconstrained by Essential 4 are the choices of representations for artifacts (SA/SD, UML, MBASE, formal specs, programming languages, …).

Example: Risk of Precise Specification

One editor specification required that every operation be available through a button on the window. As a result, the space available for viewing and editing became unusably small. The developer was precluded from moving some operations to menus because the GUI layout had been specified precisely at an early step. (Of course, given too much freedom programmers can develop very bad GUIs. Stakeholder review and prototype exercises are necessary to avoid such problems.)

Hazardous Spiral Look-Alikes: Insistence on Complete Specifications

It is often risky to undertake a spiral development project wherein complete specifications are pre-specified for *all* aspects. Aspects such as those described in the example should be left to be further defined during project exploratory phases.

Spiral Essential 5: Use Anchor Point Milestones LCO, LCA, IOC

A major difficulty of the original spiral model was its lack of intermediate milestones to serve as commitment points and progress checkpoints. This difficulty has been remedied by the development of a set of anchor point milestones:

LCO - Life Cycle Objectives - what should the system accomplish?

LCA - Life Cycle Architecture - what is the structure of the system?

IOC - Initial Operating Capability - the first released version.

(The artifacts for each are provided by an electronic process guide [10] and are also used by the Rational Unified Process.)

The focus of the LCO review is to ensure that at least one architecture choice is viable from a business perspective. The focus of the LCA review is to commit to a single detailed definition of the project. The project must have either eliminated all significant risks or put in place an acceptable risk-management plan. The LCA milestone is particularly important, as its pass/fail criteria enables stakeholders to hold up projects attempting to proceed into evolutionary or incremental development without a life cycle architecture. Each milestone is a stakeholder commitment point: at LCO the stakeholders commit to support building architecture; at LCA they commit to support initial deployment; at IOC they commit to support operations. Together the anchor point milestones avoid analysis paralysis, unrealistic expectations, requirements creep, architectural drift, COTS shortfalls and incompatibilities, unsustainable architectures, traumatic cutovers, and useless systems.

Variants: One appropriate variant of Essential 5 is the number of spiral cycles between anchor points. Another possible variant is to merge anchor points. In particular, a project using a mature and appropriately scalable fourth generation language (4GL) or product line framework will have already determined its architecture by its LCO milestone, enabling the LCO and LCA milestones to be merged.

Example: Stud Poker Analogy

A valuable aspect of the spiral model is its ability to support incremental commitment of corporate resources rather than requiring a large outlay of resources to the project before its success prospects are well understood. Funding a spiral development can thus be likened to the game of stud poker. In that game, you put a couple of chips in the pot and receive two cards, one hidden and one exposed. If your cards don't promise a winning outcome, you can drop out without a great loss. This corresponds to cancelling a project at or before LCO. If your two cards are both aces, you will probably bet on your prospects aggressively (or less so if you see aces among other players' exposed cards). Dropping out of the second or third round of betting corresponds to cancelling at or before LCA. In any case, based on information available, you can decide during each round whether it's worth putting more chips in the pot to buy more information or whether it's better not to pursue this particular deal or project.

Hazardous Look-Alike:
Evolutionary Development Without Life Cycle Architecture

The LCO and LCA milestones' pass-fail criteria emphasize that the system's architecture must support not just the initial increment's requirements, but also the system's evolutionary life-cycle requirements. This avoids the hazardous spiral look-alike of an initial increment optimized to provide an impressive early system demonstration or limited release, but without the architecture to support full-system requirements for security, fault-tolerance, or scalability to large workloads. Other important considerations for LCA are that the initial release ensure continued key stakeholder participation, that the user organizations are flexible enough to adapt to the pace of system evolution, and that legacy-system replacement be well thought out. Ignoring these aspects lead to other hazardous spiral look-alike processes.

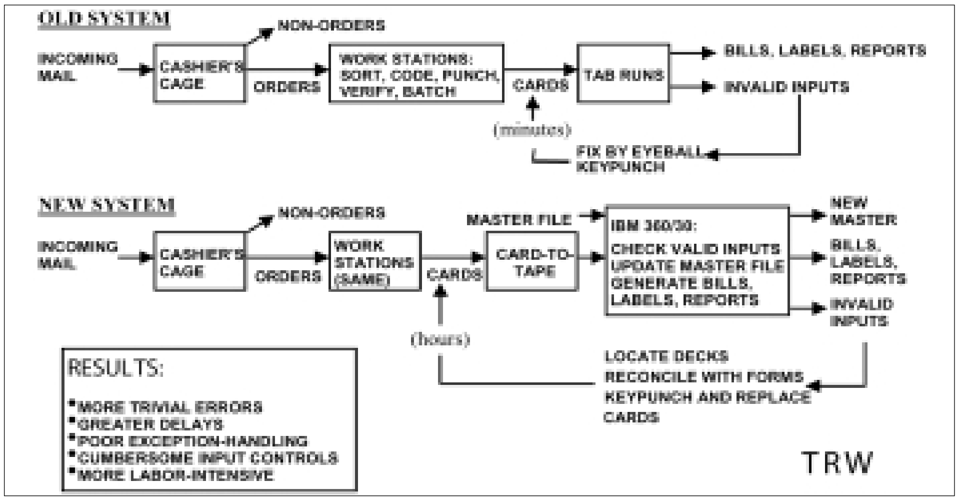


Figure 5: *Scientific American Order Processing*

Spiral Essential 6: Emphasis on System and Life Cycle Activities and Artifacts

Spiral Essential 6 emphasizes that spiral development of software-intensive systems needs to focus not just on software construction aspects, but also on overall system and life-cycle concerns. Will the product satisfy stakeholders? Will it meet cost and performance goals? Will it integrate with existing business practices? Will it adapt to organizational changes? Software developers are apt to fall into the oft-cited trap: "If your best tool is a hammer, the world you see is a collection of nails." Writing code may be a developer's forte, but it is as important to the project as are nails to a house.

Variants: The model's use of risk considerations to drive solutions makes it possible to tailor each spiral cycle to whatever mix of software and hardware, choice of capabilities, or degree of productization is appropriate.

Example: Order Processing

An example of failure to consider the whole system occurred with the Scientific American order processing system in Figure 5 [11]. Scientific American had hoped that computerizing the functions being performed on tabulator machines would reduce its subscription processing costs, errors, and delays. Rather than analyze the sources of these problems, the software house focused on the part of the problem having a software solution. The result was a batch-processing computer system whose long delays put extra strain on the clerical portion of the system that had been the major source of costs, errors, and delays in the first place. The software people looked for the part of the problem with a software solution (their "nail"), pounded it in with their software hammer, and left Scientific American worse off than when they started.

This kind of outcome would have resulted even if the software automating the tabulator-machine functions had been developed in a risk-driven cyclic approach. However, its Life Cycle Objectives milestone package would have failed its feasibility review, as it had no system-level business case demonstrating that the development of the software would lead to the desired reduction in costs, errors, and delays. Had a thorough business case analysis been done, it would have identified the need to reengineer the clerical business processes as well as to automate the manual tab runs.

Hazardous Spiral Look-Alikes:
Logic-Only OO Designs

Models excluded by Essential 6 include most published object-oriented analysis and design (OOA&D) methods, which are usually presented as abstract logical exercises independent of system performance or economic concerns. For example, in a recent survey of 16 OOA&D books, only six listed the word "performance" in their index, and only two listed "cost."

Using the Spiral Model for Evolutionary Acquisition

Both the February and September workshops had working groups on the relationships between spiral development and evolutionary acquisition. A primary conclusion was that the relationships differ across two major DoD acquisition sectors:

- Information systems, such as C4ISR systems, logistics systems, and management systems, in which spiral and evolutionary models coincide well.
- Software-intensive embedded hardware/software systems, in which the software aspects best follow a spiral approach, but the hardware aspects need to follow a more sequential approach to accommodate lead times for production facilities, production subcontracts, and long-lead critical component orders.

Even for embedded systems, however, spiral approaches can be helpful for synchronizing hardware and software processes, and for determining when to apply an evolutionary, incremental, or single-step acquisition strategy. For example, Xerox's time-to-market process uses the spiral anchor point milestones as hardware/software synchronization points for its printer business line [12]. Rechtin and Maier adopt a similar approach in their book, the Art of Systems Architecting [13].

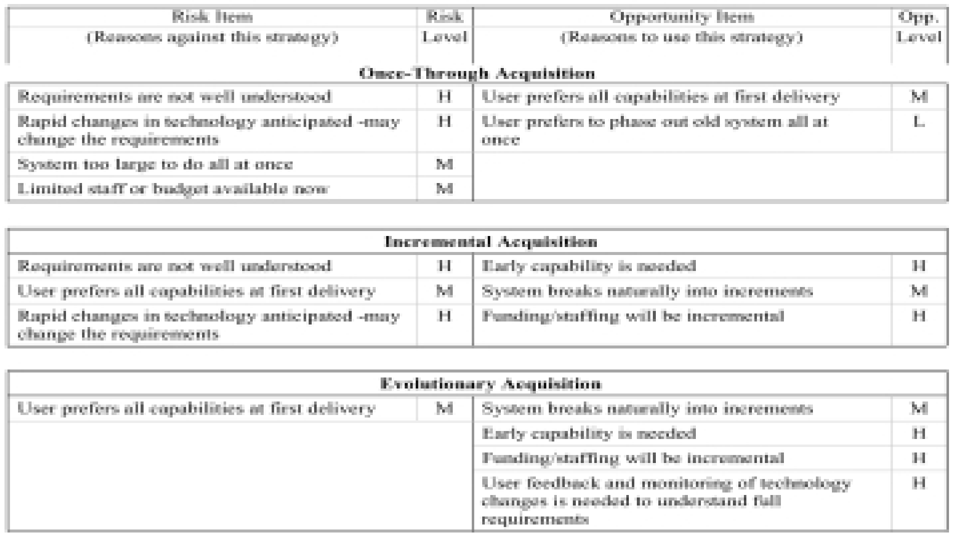
Development strategy	Define all requirements first?	Multiple development cycles?	Distribute interim software?
Once-Through (Waterfall)	Yes	No	No
Incremental (Preplanned Product Improvement)	Yes	Yes	Maybe
Evolutionary	No	Yes	Yes

Table 1: *Evolutionary Acquisition Distinctions [14]*

A good example of the use of a risk-driven spiral approach to determine a preferred software/system acquisition strategy was originally developed for DoD's MIL-STD-498, and subsequently incorporated in IEEE/EIA 12207 [14]. This approach distinguishes among single-step (once-through or waterfall), incremental, and evolutionary acquisition processes as shown in Table 1. Thus, evolutionary acquisition avoids defining all requirements first, and proceeds in multiple development cycles, some of which involve distribution and usage of initial and intermediate operational capabilities.

Table 2 shows how a spiral risk analysis can be performed during early integrated product and process definition cycles to select the most appropriate acquisition process for a system. The example shown in Table 2 is for a fairly large, high-technology C4ISR system. For such a system, the high risks of poorly-understood requirements and rapid technology changes push the decision away from once-through or incremental acquisition, while the needs for an early capability and for user feedback on full requirements push the decision toward evolutionary acquisition. If the system had been a small, lower-technology embedded system where all capabilities are needed at first delivery, the risk and opportunity factors would push the decision away from evolutionary and incremental acquisition toward once-through or single-step acquisition.

Table 2: *Spiral Risk Analysis for Process Selection [14]*



Risk Item (Reasons against this strategy)	Risk Level	Opportunity Item (Reasons to use this strategy)	Opp. Level
Once-Through Acquisition			
Requirements are not well understood	H	User prefers all capabilities at first delivery	M
Rapid changes in technology anticipated -may change the requirements	H	User prefers to phase out old system all at once	L
System too large to do all at once	M		
Limited staff or budget available now	M		
Incremental Acquisition			
Requirements are not well understood	H	Early capability is needed	H
User prefers all capabilities at first delivery	M	System breaks naturally into increments	M
Rapid changes in technology anticipated -may change the requirements	H	Funding/staffing will be incremental	H
Evolutionary Acquisition			
User prefers all capabilities at first delivery	M	System breaks naturally into increments	M
		Early capability is needed	H
		Funding/staffing will be incremental	H
		User feedback and monitoring of technology changes is needed to understand full requirements	H

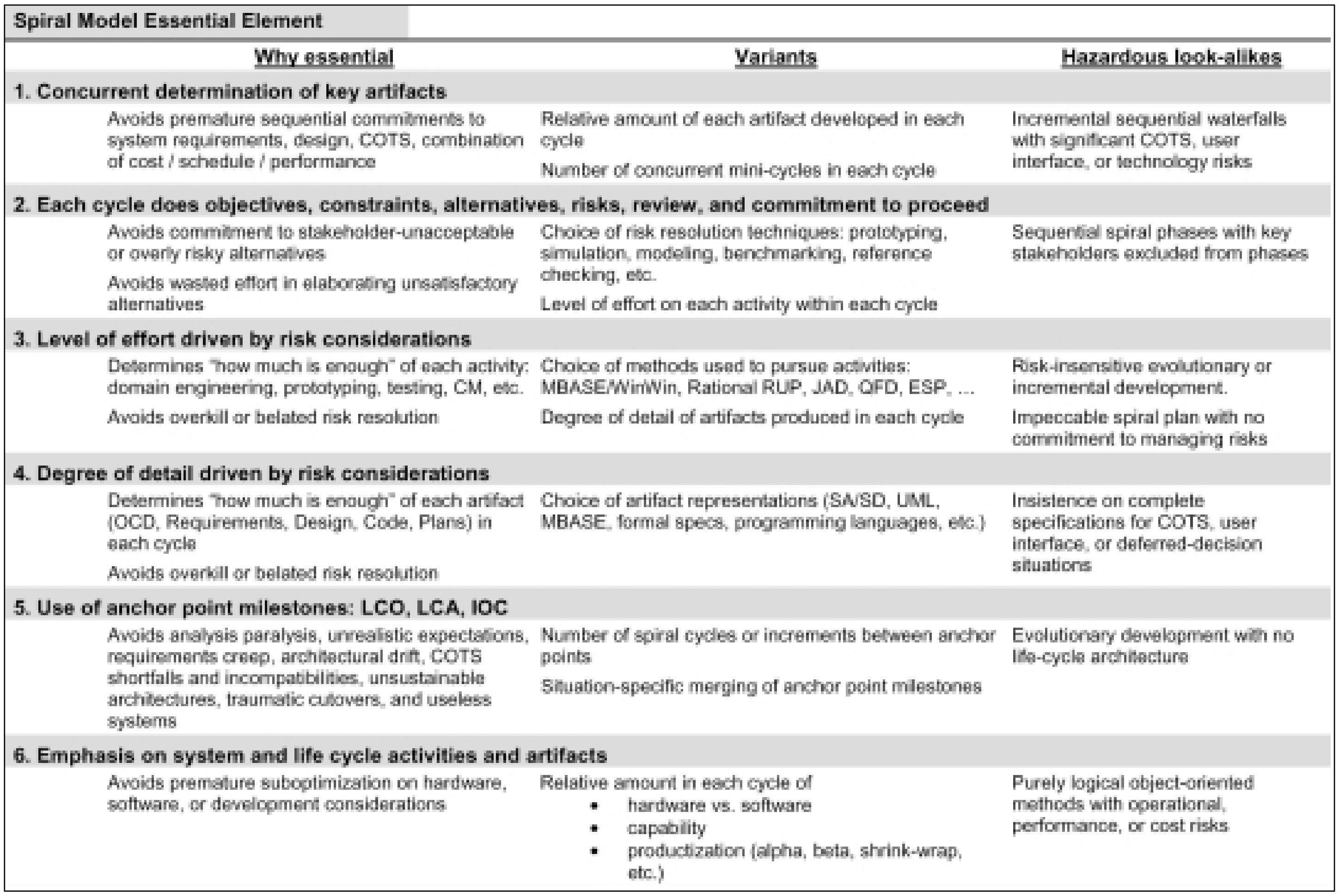
Conclusion

This paper has defined the spiral development model as a risk-driven process model generator with cyclic process execution and a set of three anchor point milestones. The definition was sharpened by presenting a set of six "essential" attributes; that is, six attributes which every spiral development process must incorporate. These essentials are summarized in Table 3 (See page 10). Omission of each of these gives rise to process models that are cyclic or iterative, but are not examples of spiral development. These are called "hazardous spiral look-alikes." Each was described and pilloried as part of describing the Essential it violates.

Spiral development works fairly seamlessly with evolutionary acquisition of information systems. For evolutionary acquisition of software-intensive embedded hardware-software systems, a mix of spiral and sequential processes is often needed. Even here, however, the spiral anchor points and risk-driven process selection approach are useful for determining how best to synchronize the hardware and software processes.◆

References

1. Boehm, B., A Spiral Model of Software Development and Enhancement, *Computer,* May 1988, pp. 61-72.
2. Department of Defense Instruction 5000.2, Operation of the Defense



Spiral Model Essential Element	Why essential	Variants	Hazardous look-alikes
1. Concurrent determination of key artifacts			
	Avoids premature sequential commitments to system requirements, design, COTS, combination of cost / schedule / performance	Relative amount of each artifact developed in each cycle Number of concurrent mini-cycles in each cycle	Incremental sequential waterfalls with significant COTS, user interface, or technology risks
2. Each cycle does objectives, constraints, alternatives, risks, review, and commitment to proceed			
	Avoids commitment to stakeholder-unacceptable or overly risky alternatives Avoids wasted effort in elaborating unsatisfactory alternatives	Choice of risk resolution techniques: prototyping, simulation, modeling, benchmarking, reference checking, etc. Level of effort on each activity within each cycle	Sequential spiral phases with key stakeholders excluded from phases
3. Level of effort driven by risk considerations			
	Determines "how much is enough" of each activity: domain engineering, prototyping, testing, CM, etc. Avoids overkill or belated risk resolution	Choice of methods used to pursue activities: MBASE/WinWin, Rational RUP, JAD, QFD, ESP, ... Degree of detail of artifacts produced in each cycle	Risk-insensitive evolutionary or incremental development. Impeccable spiral plan with no commitment to managing risks
4. Degree of detail driven by risk considerations			
	Determines "how much is enough" of each artifact (OCD, Requirements, Design, Code, Plans) in each cycle Avoids overkill or belated risk resolution	Choice of artifact representations (SA/SD, UML, MBASE, formal specs, programming languages, etc.)	Insistence on complete specifications for COTS, user interface, or deferred-decision situations
5. Use of anchor point milestones: LCO, LCA, IOC			
	Avoids analysis paralysis, unrealistic expectations, requirements creep, architectural drift, COTS shortfalls and incompatibilities, unsustainable architectures, traumatic cutovers, and useless systems	Number of spiral cycles or increments between anchor points Situation-specific merging of anchor point milestones	Evolutionary development with no life-cycle architecture
6. Emphasis on system and life cycle activities and artifacts			
	Avoids premature suboptimization on hardware, software, or development considerations	Relative amount in each cycle of • hardware vs. software • capability • productization (alpha, beta, shrink-wrap, etc.)	Purely logical object-oriented methods with operational, performance, or cost risks

Table 3: *Summary*

Acquisition System, September 2000, www.acq.osd.mil/ap/i50002p.doc

3. Hansen, W.; Foreman, J.; Carney, D; Forrester, E.; Graettinger, C.; Peterson, W.; and Place, P., Spiral Development Building the Culture: A Report on the CSE-SEI Workshop, February 2000, Software Engineering Institute, Carnegie Mellon University, Special Report CMU/SEI-2000-SR-006, July 2000, www/cbs/spiral2000/february2000/finalreport.html
4. Hansen, W.; Foreman, J.; Albert, C.; Axelband, E.; Brownsword, L.; Forrester, E. ; and Place, P., Spiral Development and Evolutionary Acquisition: The SEI-CSE Workshop, September, 2000, Software Engineering Institute, Carnegie Mellon University, Special Report, in preparation.
5. Boehm, Barry, edited by Hansen, Wilfred J., Spiral Development: Experience, Principles, and Refinements, Software Engineering Institute, Carnegie Mellon University, Special Report CMU/SEI-00-SR-08, ESC-SR-00-08, June, 2000, www/cbs/ spiral2000/february2000/BoehmSR.html.
6. Carr, M. J.; Konda, S. L.; Monarch, I.; Ulrich, F. C., and Walker, C. F., Taxonomy-Based Risk Identification, Software Engineering Institute, Carnegie Mellon University, Technical Report CMU/SEI-93-TR-6, ESC-TR-93-183, June,1993,www.sei.cmu.edu/legacy/risk/kit/tr06.93.pdf
7. Williams, R. C.; Pandelios, G. J.; and Behrens, S.G., Software Risk Evaluation (SRE) Method Description (Version 2.0), Software Engineering Institute, Carnegie Mellon University, Technical Report CMU/SEI-99-TR-029, ESC-TR-99-029, December, 1999, www.sei.cmu.edu/pub/documents/99.reports/pdf/99tr029body.pdf
8. Boehm, B., Unifying Software Engineering and Systems Engineering, *IEEE Computer*, March 2000, pp. 114-116.
9. Boehm, B., *Software Risk Management*, IEEE Computer Society Press, 1989.
10. Mehta, N., *MBASE Electronic Process Guide*, USC-CSE, Los Angeles, Calif., October 1999, sunset.usc.edu/research/MBASE/EPG
11. Boehm, B., *Software Engineering Economics*, New York, N.Y., Prentice Hall, 1981.
12. Hantos, P., From Spiral to Anchored Processes: A Wild Ride in Lifecycle Building architecture, Proceedings, USC-SEI Spiral Experience Workshop, Los Angeles, Calif., Febuary 2000, www.sei.cmu.edu/cbs/spiral2000/Hantos
13. Rechtin, E., and Maier, M., *The Art of Systems Architecting*, CRC Press, 1997.
14. IEEE and EIA, Industry Implementation of ISO/IEC 12207: Software Life Cycle Processes-Implementation Considerations, IEEE/EIA 12207.2 - 1997, April 1998.

Article Web Site Location

http://www.sei.cmu.edu/pub/documents/00.reports/pdf/00sr008.pdf

"It is impossible to make a program foolproof because fools are so ingenious."
Anonymous

About the Authors

Barry Boehm, Ph.D., is the TRW professor of software engineering and director of the Center for Software Engineering at the University of Southern California. He was previously in technical and management positions at General Dynamics, Rand Corp., TRW, and the Office of the Secretary of Defense as the director of Defense Research and Engineering Software and Computer Technology Office. Boehm orginiated the spiral model, the Constructive Cost Model, and the stakeholder win-win approach to software management and requirements negotiation.

University of Southern California
Center for Software Engineering
Los Angeles, CA 90089-0781
Voice: 213-740-8163
Fax: 213-740-4927
E-mail: boehm@sunset.usc.edu

Wilfred J.Hansen has been consulting on the tools and processes for utilizing commercial-off-the-shelf (COTS) software within larger systems at the Software Engineering Institute. Previously he was director of the Andrew Consortium where he led the development and maintenance of the Andrew User Interface System, which was the first modern word-processor in that it originated the embedding of arbitrary objects into text and into other objects. One of his contributions to Andrew was a scripting language having a unique integer-free sub-language for processing strings. Earlier in his career, Hansen taught data structures and programming languages.

Software Engineering Institute
4500 Fifth Avenue
Pittsburgh, PA 15213-3890
Voice: 412-268-8247
E-mail: wjh@sei.cmu.edu

Web sites

Software Technology Support Center

www.stsc.hill.af.mil

The Software Technology Support Center (STSC) was established in 1987 as the command focus for proactive application of software technology in weapon, command and control, intelligence, and mission-critical systems. The STSC provides hands-on assistance in adopting effective technologies for software-intensive systems. It helps organizations identify, evaluate, and adopt technologies that improve software product quality, production efficiency, and predictability. STSC uses the term technology in its broadest sense to include processes, methods, techniques, and tools that enhance human capability. Its focus is on field-proven technologies that will benefit the Department of Defense mission.

Software Engineering Institute

www.sei.cmu.edu

The Software Engineering Institute (SEI) is a federally funded research and development center sponsored by the Department of Defense to provide leadership in advancing the state of the practice of software engineering to improve the quality of systems that depend on software. SEI helps organizations and individuals to improve their software engineering management practices. The site features a software engineering management practices work area that focuses on the ability of organizations to predict and control quality, schedule, cost, cycle time, and productivity when acquiring, building, or enhancing software systems.

Data & Analysis Center for Software

www.dacs.dtic.mil

The Data and Analysis Center for Software (DACS) is a Department of Defense (DoD) Information Analysis Center. The DACS is the DoD software information clearinghouse serving as an authoritative source for state of the art software information and providing technical support to the software community. The center's technical area of focus is software technology and software engineering, in its broadest sense. The DACS offers a wide variety of technical services designed to support the development, testing, validation, and transitioning of software engineering technology. The DACS is technically managed by the Air Force Research Laboratory - Information Directorate.

The Institute of Electrical and Electronics Engineers

www.ieee.org

The Institute of Electrical and Electronics Engineers, Inc. (IEEE), helps advance global prosperity by promoting the engineering process of creating, developing, integrating, sharing, and applying knowledge about electrical and information technologies and sciences for the benefit of humanity and the profession. The IEEE is a nonprofit, technical professional association of more than 350,000 individual members in 150 countries. Through its members, the IEEE is a leading authority in technical areas ranging from computer engineering, biomedical technology and telecommunications, to electric power, aerospace and consumer electronics, among others.

Software Productivity Consortium

www.software.org

The Software Productivity Consortium is a unique, nonprofit partnership of industry, government, and academia. It develops processes, methods, tools, and supporting services to help members and affiliates build high-quality, component-based systems, and continuously advance their systems and software engineering maturity pursuant to the guidelines of all of the major process and quality frameworks. The site features an interactive section to discuss new trends.

Verification and Validation Implementation at NASA

Dr. Linda H. Rosenberg
NASA, Goddard Space Flight Center

Any company that produces or relies on software knows the importance of high quality and reliability. Recently NASA focused on applying Independent Verification and Validation (IV&V) as part of a software improvement effort. This paper is not about IV&V, but about NASA's new IV&V implementation approach on all software development throughout the agency. This information is valuable to any project, organization, or company considering applying IV&V.

NASA is recognized as a leader in space technology, using cutting-edge science to probe galaxies never before seen by mankind. In keeping with this cutting-edge technology, much of the functionality previously done through hardware has transferred to software, including mission critical functions. But technology implementation is moving so fast, that at times quality assurance cannot keep up, although we try. The NASA Independent Verification and Validation (IV&V) Facility was established in 1993 in West Virginia and tasked to support NASA projects in achieving the highest levels of safety and cost effectiveness for mission-critical software. Despite this, NASA has experienced some failures recently that were traced, in part, to less than adequate implementation of mission software.

NASA determined the need for software IV&V after evaluating the causes of recent mission failures. These were due, in part, to software issues that should have been identified during development or testing. The NASA IV&V Facility was developed to be a center of excellence, but was underutilized. Projects that did use the facility had proven benefits. Thus NASA's focus for improvement includes increased software IV&V application.

NASA began by looking at available resources and projects that had applied IV&V to determine the benefits in the NASA environment. It was quickly determined that only a few projects were implementing IV&V; not all were taking advantage of the facility's expertise. There were very definite proven benefits to NASA when IV&V was applied. These ranged from cost savings to identifying mission critical errors not previously identified through testing. Applying IV&V on this software resulted in increased safety and mission reliability. However, an identified deficiency was that there was no consistent application of IV&V by the projects.

This paper discusses the approach taken to increase the use of IV&V within NASA. We will start by defining independent, verification, and validation. We will then discuss the written policy relative to the performance of software IV&V, and the criteria developed to help projects quantify the need for IV&V.

Independent Verification and Validation

A basic understanding of what constitutes IV&V begins with the definitions of verification and validation, then determines what is required for "I" – independence. The Institute of Electrical and Electronics Engineers (IEEE) 610.12-1990, Standard Glossary of Software Engineering Terminology, defines verification and validation [1]. Verification is defined as the process of evaluating a system or component to determine whether the products of a given development phase satisfy the conditions imposed at the start of that phase. Validation is defined as the process of evaluating a system or component during or at the end of the development process to determine whether it satisfies specified requirements. Verification asks "Did we build the system right?" and validation asks "Did we build the right system?"

IEEE defines *independence* in IV&V as three parameters: technical independence, managerial independence, and financial independence. Personnel who are not involved in software development achieve technical independence. IV&V personnel use their expertise to assess development processes and products independent of the developer. They formulate their own understanding of potential problems and how the proposed system is solving them.

Managerial independence requires responsibility for the IV&V effort to be vested in an organization separate from the organization responsible for performing the system implementation. The IV&V effort independently selects the segments of the software and system to analyze and test, chooses the IV&V techniques, defines the schedule of IV&V activities, and selects the specific technical issues and problems to act upon. Most projects view V&V as sufficient and do not recognize the added value the independence brings.

Financial independence has been harder to attain. All work on the project, including quality assurance, is funded directly by the project; hence, IV&V is also funded directly by the project. In theory, the project team could remove IV&V funding if they are not satisfied with the findings. But with the implementation of the IV&V policy, projects are now required to work with the IV&V Facility to reach an agreement on the amount of IV&V and funding. The center director must agree to any changes to this along with strong justification by the project manager.

Implementation Approach

All software project managers, whether government or industry, never have time or money to spare, so when NASA identified the need for IV&V, it was met with a cry of, "Not on my project!" But the implementation of IV&V was part of a larger effort to improve the software developed at NASA, to achieve the highest levels of safety and cost effectiveness possible for mission critical software. The NASA IV&V Facility provides tailored technical, project management and financial analysis for NASA projects, industry, and other government agencies, by applying software engineering "best practices" to evaluate software risk and criticality throughout the system development life cycle. Through the facility, NASA has the means for implementing IV&V, but it is under utilized. Only a few very large NASA projects, such as Space Station, chose to apply IV&V through the facility.

To change this, the first step was to write a policy requiring projects to investigate the necessity of doing IV&V. It is an effective risk mitigation strategy, and since most NASA missions are cutting edge technology, they also are at high risk. Recognizing that cost must be balanced against potential benefits, the "amount" of risk incurred by a project had to be assessed. A policy was developed that included the process of determining the need for IV&V, the extent, and approach. The policy also states all IV&V will be done under the management of the NASA IV&V Facility, centralizing expertise and ensuring consistency.

The next step was to develop criteria determining when to consider IV&V. This required quantifying project risk for an initial assessment on which projects may require IV&V. Project risk is defined as a combination of the probability that an undesirable event will occur, and the consequence if the event does occur. The IV&V criteria were written using probability and consequence. Factors influencing software development were identified, and risk factors associated with them for a calculation of the probability. Consequences of failure were classified as Grave, Substantial, Marginal, and Insignificant. These are combined for a determination of the necessity of IV&V.

The final step, and the hardest in some respects, was to identify the projects that potentially required IV&V and to determine to what level IV&V was needed. Money is always an issue; there is never enough in any software development. So what was the cost, and what are the balancing benefits?

The approach to implement IV&V consistently and logically on all NASA software was broken into three steps:

1. Write a policy for the requirement of IV&V implementation.
2. Write the criteria for an initial determination of IV&V necessity.
3. Work with projects to implement IV&V.

Step 1: IV&V Implementation Policy

The policy was to clearly specify the process of determining when a program must apply IV&V under the management of the NASA IV&V Facility. One strength of the policy is the specification that the NASA IV&V Facility is responsible for the management of all software IV&V efforts within the agency. This creates a central repository of knowledge, tools, metrics, and lessons learned that can be used to improve the IV&V efforts on future projects throughout NASA.

The policy states that each project must produce, document, and implement a plan that addresses V&V performance; and if appropriate, IV&V through the software life cycle – from requirements through delivery and maintenance. The level of IV&V of software that is performed is based on the cost, size, complexity, life span, risk, and consequences of failure as defined using the criteria explained in the following section.

Step 2: IV&V Criteria

In order to accomplish the goal to increase the application of IV&V on all appropriate projects, it had to be determined what was meant by "appropriate" projects, hence the development of IV&V criteria. Looking back on the IV&V objective for risk mitigation, those projects with high risk had to be identified; but first, the criteria for what makes a software project high risk had to be defined and quantified. The quantification was the hardest part.

IV&V is intended to assist mitigating risk, hence the decision to do IV&V must be risk-based. NASA policy defines risk as the "combination of 1) the probability (qualitative or quantitative) that a program or project will experience an undesired event such as cost overrun, schedule slippage, safety mishap, or failure to achieve a needed breakthrough; and 2) the consequences, impact, or severity of the undesired event were it to occur [3]." The likelihood of occurrence and consequences of a given software failure cannot be calculated early in the software life cycle. However, there are realistic metrics available that give good general approximations of the consequences as well as the likelihood of failures.

Probability Evaluation

The probability of failure for software is difficult to determine at any phase in the software development life cycle. NASA has identified factors that impact development difficulty. These factors were then calibrated to determine the extent of risk for successful software development. While the indicators are not precise and are currently in Beta testing by NASA software development projects and the IV&V determination team, they are available to provide estimates that are adequate for assessing IV&V need.[1] There are nine factors: software team complexity, contractor support, organization complexity, schedule pressure, process maturity of software provider, degree of innovation, level of integration, requirement maturity, and software lines of code.

Five risk categories within each factor were identified based on input from software developers from all NASA centers. Values 1, 2, 4, 8, and 16 were assigned to each category. Finally, a weighting factor of 1 or 2 was identified for each factor. This information is shown in Table 1

To apply the criteria, the project manager identifies the category of risk for each factor and multiplies the appropriate value (1, 2, 4, 8, or 16) times the weighting factor of 1 or 2. The sum for all factors yields an initial numerical representation of the project software development risk. For example, a project might rate the following:

- Software team complexity – "up to 20 people at one location" = 4 * 2 = 8
- Contractor support – "with minor tasks" = 2 * 2 = 4
- Organizational complexity – "two locations but with same reporting chain" = 2 * 1 = 2
- Schedule pressure – "non-negotiable" = 16 * 2 = 32
- Process maturity – " CMM Level 1 but with a successful history" = 8 * 2 = 16
- Innovation – "between proven but new and cutting edge" = 8 * 1 = 8
- Integration – "almost stand alone" = 2 * 2 = 4
- Requirement maturity – "preliminary objectives" = 8 * 2 = 16
- Lines of code – "~ 300K" = 2 * 2 = 4
- TOTAL = 8+4+2+32+16+8+4 + 16+4= 94

This risk information must now be combined with the consequence evaluation.

Factors contributing to probability of software failure	Un-weighted probability of failure score					Weighting Factor	Likely-hood of failure rating
	1	2	4	8	16		
Software team complexity	Up to 5 people at one location	Up to 10 people at one location	Up to 20 people at one location or 10 people with external support	Up to 50 people at one location or 20 people with external support	More than 50 people at one location or 20 people with external support	X2	
Contractor Support	None	Contractor with minor tasks		Contractor with major tasks	Contractor with major tasks critical to project success	X2	
Organization Complexity*	One location	Two locations but same reporting chain	Multiple locations but same reporting chain	Multiple providers with prime sub relationship	Multiple providers with associate relationship	X1	
Schedule Pressure*	No deadline		Deadline is negotiable		Non-negotiable deadline	X2	
Process Maturity of Software Provider	Independent assessment of Capability Maturity Model (CMM) Level 4, 5	Independent assessment of CMM Level 3	Independent assessment of CMM Level 2	CMM Level 1 with record of repeated mission success	CMM Level 1 or equivalent	X2	
Degree of Innovation	Proven and accepted		Proven but new to the development organization		Cutting edge	X1	
Level of Integration	Simple - Stand alone				Extensive Integration Required	X2	
Requirement Maturity	Well defined objectives - No unknowns	Well defined objectives - Few unknowns		Preliminary objectives	Changing, ambiguous, or untestable objectives	X2	
Software Lines of Code*	Less than 50K		Over 500K		Over 1000K	X2	
Total							

Table 1: *Likelihood of Failure Based on Software Environment* (* indicates additional information is also required)

Consequence Evaluation

In general, the consequences of a software failure can be derived from the purpose of the software: i.e., what does the software control; what do we depend on it to do? NASA has many types of software, including flight software that is launched and contains mission critical functionality, ground system software that sends commands, scientific software for the experiments, and just about all other types of software imaginable. There are factors that can be used to categorize software based on its intended function as well as the level of effort expended to produce it: potential for loss of life, potential for serious injury, potential for catastrophic mission failure, potential for partial mission failure, potential for loss of equipment, potential for waste of software resource investment, potential for adverse visibility, and potential effect on routine operations.

Now the potential for failure had to be quantified. Four ratings were chosen: Grave, Substantial, Marginal and Insignificant. Each of the factors above were quantified for each rating. If *any* of the conditions are met, the software is considered to reside in that category. For example, the category Grave is defined as follows:

- Potential for loss of life - Yes.
- Potential for loss of equipment – Greater than $100,000,000.
- Potential for waste of resource investment – Greater than 200 work years on software.
- Potential for adverse visibility – International.

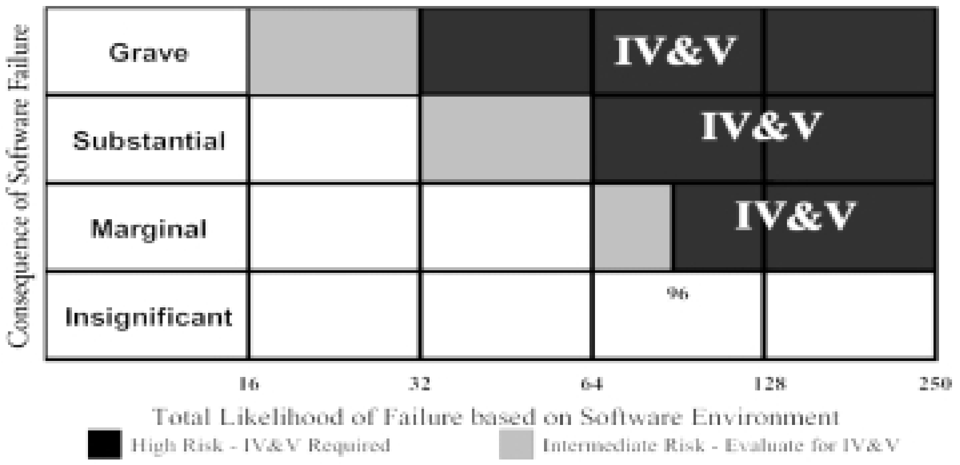
Combining the results of the "likelihood of failure rating" and the "consequences of failure" yields a risk assessment that can be used to identify the need for IV&V. Applying these criteria only determines that a project is an IV&V candidate – not the level of IV&V nor the resources associated with the IV&V effort. These must be determined as a result of discussions between the project and the IV&V Facility. Figure 1 shows a dark region of high risk where software consequences, likelihood of failure, or both are high. Projects having software that falls into this high-risk area shall undergo IV&V. The exception is those projects that have already done hardware/software integration. An IV&V would not be productive that late in the development cycle. The gray regions represent projects with intermediate risk. Projects having software that falls into these areas shall undergo an evaluation to determine if IV&V is warranted. Using the previous project example, a probability of 94 score means that they must perform IV&V if the consequence of failure is Substantial or Grave. Since the project's consequence was determined to be Grave, they must perform IV&V under the NASA IV&V Facility.

Step 3: Project Implementation

Prior to implementing IV&V, a company has to know about the software it develops. At NASA, software development is done at all 10 centers spread from California to Florida, Texas to Ohio, and Alabama to Maryland, and with universities and industries. In total, criteria data were received on approximately 100 projects. After discussions with projects to clarify the information and correct erroneous data, approximately 70 projects were identified as IV&V candidates.

Applying IV&V to 70 projects immediately, however, is impossible. The facility currently does not have the resources to accommodate this many projects and this

Figure 1: *Pre-Ship Test Risk Exposure*



much work at this time. Using the data from the criteria (consequence and probability) and discussions with the project managers, the following guidelines to focus the IV&V efforts were implemented:

- All projects currently receiving IV&V will continue.
- All projects classified Grave should be addressed first for IV&V with the highest priority applied to those closest to operational date as a general rule, with some attention applied to why it is classified as Grave.
- All projects classified as Substantial and needing IV&V based on the risk probability value greater than 32 that are in the requirements or design phase.
- All remaining projects classified as Substantial and needing IV&V based on the risk probability value greater than 32.
- All remaining projects classified as Marginal and needing IV&V based on the risk probability value greater than 96, prioritized based upon how close to starting they are.

Results

Applying IV&V on NASA projects has shown some very positive results and prevented costly errors. In one project, the IV&V activity identified design flaws in the command-and-control system that, if not corrected, would have resulted in a catastrophic hazard. This critical piece of software sends commands to hardware elements, and if not working properly, could fail to send emergency response commands leading to the loss of attitude control, rapid depressurization, and other hazardous conditions. Using a code analysis, IV&V identified an error that would eliminate the vital command link between the ground control system and the satellite. A special software patch was generated for the on-orbit software to correct the problem. IV&V developed policy criteria used by one program for non-flight software in integrated tests. This policy was key for insuring testing integrity while making it possible to keep tight development schedules. IV&V activities have benefited many different domains of NASA's software. In the manned-space flight domain, more than 4,000 problems were identified, 10 of the highest criticality, those that could result in loss of mission or loss of life. For experimental flight vehicles, IV&V identified more than 300 requirements and design problems. For ground systems, more than 250 legacy system requirements and mitigation problems were identified.These are just some of the benefits NASA has reaped from the formally applying IV&V. We currently are working on the return-on-investment calculations for these activities.

Conclusion

The results of NASA's investigation have shown that in this environment, IV&V has been cost effective. Companies large and small, in today's competitive world cannot afford software that is unreliable. To be effective, however, IV&V must be applied effectively, and the independence must not be lost. Project managers must understand the benefits above the cost, looking at the whole development, not just the current state.

NASA has recognized the value of IV&V and has taken steps to implement IV&V on all software projects where warranted. The decision to implement IV&V is no longer solely the decision of the project manager, but through an independent evaluation, the risk of the project is evaluated, and the need for IV&V is determined. Although the policy and criteria in this paper were written for NASA projects, they are applicable with minor modification to any software development.◆

Acknowledgements

Much of the work presented in this paper was a combined effort of NASA IV&V Facility personnel (Judy Bruner, John Hinkle, and Ken McGill), Goddard Space Flight Center personnel (Charles Vanek, John Dalton, Linda Rosenberg), and the members of the Software Working Group under Pat Schuller, Langley Research Center.

References

1. IEEE 610.12-1990, *IEEE Standard Glossary of Software Engineering Terminology*, Institute of Electrical and Electronics Engineers, Inc.
2. NASA IV&V Business Plan, Office of System Safety and Mission Assurance, NASA Goddard Space Flight Center, MD , June 2000, ivvplan.gsfc.nasa.gov
3. NASA Policy Guideline (NPG) 7120.5, NASA Program and Project Management Processes and Requirements Highlight Code, Office of Chief Engineer, NASA Headquarters, Washington, DC.

Note

1. Initial beta testing results indicate that some values and weights need to be adjusted. This will be done in the next release this year.

About the Author

Linda Rosenberg, Ph.D., serves as the chief scientist for Software Assurance at NASA's Goddard Space Flight Center, and is the former division chief of the Software Assurance Technology Office (SATO). Dr. Rosenberg is a recognized international expert in the areas of software assurance, software metrics, requirements and reliability. She serves on the Institute of Electrical and Electronics Engineers program committees for software reliability, software metrics, and software requirements. She is also an adjunct professor at the University of Maryland, Baltimore. Dr. Rosenberg holds a Ph.D. in computer science from the University of Maryland, a M.E.S. in computer science, and a batchelor's in mathematics.

Software Assurance Technology Office, Code 304
Goddard Space Flight Center, NASA
Greenbelt, MD 20771 USA
Voice: 301-286-0087
Fax: 301-286-1667
E-mail: Linda.Rosenberg@gsfc.nasa.gov

"Computers can figure out all kinds of problems, except the things in the world that just don't add up."
James Magary

"When we write programs that learn, it turns out that we do and they don't."
Alan J. Perlis

The Next Refinement to Software

KeithThurston
Office of Government-Wide Policy

Access for persons with disabilities has been an issue in software development for a number of years. However, the importance and impact has reached new levels with a change in the law regarding access to federal information.

To ensure that technology would be accessible to individuals with disabilities, Congress added Section 508 to Title V of the Rehabilitation Act in 1986. Under Section 508 (29 U.S.C. 794d) agencies must give disabled employees and members of the public access to information that is comparable to the access available to others. The law specifically exempted intelligence systems and military command and control systems, but applies to all other systems such as logistics and administrative systems. However, little progress toward fulfilling Section 508's objectives prompted the introduction of new legislation in 1998. It establishes an enforcement procedure to strengthen Section 508.

There are 54 million Americans with some level of disability. In the federal workforce there are 167,902 employees with reportable disabilities and 28,672 employees with targeted (significant) disabilities. The real number of Americans with disabilities is actually larger because the 97 percent of the population born without disabilities fail to report them to human resources departments when they are acquired later in life. They simply adopt methods and processes that accommodate their disability to permit them to remain productive.

Who Will Be Impacted?

The 1998 law has government-wide impact, including contractors and will essentially be society-wide as the electronics and information technology industry continues to make all products and services more accessible. Citizens with disabilities and employees will have the right to sue federal agencies for non-compliance and inaccessible information in any program area where information technology procurements have been made after June 21, 2001.

Information technology accessibility for persons with disabilities is a fascinating area that encompasses multi-modality – individuals' ability to request and retrieve information in the mode that is best suited and most convenient at the time. It is really just the leading edge of features that we all like to have as we become a more mobile workforce. There are times when I really need to access my e-mail by telephone and have the messages read to me. Cell phones that respond to voice commands while driving are the type of capability that voice recognition systems for the blind provide for the mass market.

Section 508 applies to all electronic and information technology: the Web, software and hardware, photocopy machines, audio-video, telecommunications, and kiosks. In fiscal year 1999, the federal government purchased $37.6 billion in information technology. It is estimated that $12.4 billion of that would be subject to Section 508. The regulatory analysis conducted in conjunction with the 508 final regulation estimated the added government cost of Section 508 between $85 million and $691 million annually. The legal requirements are clear: "When developing, procuring, maintaining, or using electronic and information technology, each federal department or agency, … shall ensure, unless an undue burden would be imposed on the department or agency, that the electronic and information technology allows, regardless of the type of medium of the technology:

i) individuals with disabilities who are federal employees to have access to and use of information and data that is comparable to the access to and use of the information and data by federal employees who are not individuals with disabilities; and

(ii) individuals with disabilities who are members of the public seeking information or services from a federal department or agency to have access to and use of information and data that is comparable to the access to and use of the information and data by such members of the public who are not individuals with disabilities." The regulations for Section 508 primarily targets software development issues that preclude access to information for persons with blindness or low vision, deafness or difficulty hearing, or persons with mobility or dexterity limitations.

Keyboard Equivalents

Making information totally accessible using keyboard commands provides the interface for most of the special accommodation devices that are used by persons with blindness and dexterity limitations. While this approach works well within an application, there are development considerations. Developers must consider the entire software stack and operating environment to ensure that keyboard commands in the application software do not conflict with commands operating through other software layers. Accessibility requires clear standards and consistency for reserved keyboard commands and function keys. The regulation requires that applications do not disrupt or disable features of other products that are identified as accessibility features.

Many commercial products with graphical user interfaces have very well developed keyboard equivalencies to perform functions. However, one problem is consistent keyboard commands, which really is the motif or metaphor for the person with blindness. If each application randomly chooses a different set of keyboard commands for the close and exit commands, it then places a real limit on the number of applications a person can learn and memorize. A good solution for this while organizations are thinking about standardized metaphors and motifs is to adopt a popular set of keyboard equivalents.

A second problem with keyboard commands is that some applications have commands that require three or four keys to activate. While this may work fine for a person with blindness, it will cause havoc for people with dexterity limitations. To accommodate the four-key command, they may have to slow the activation speed of their accessibility so much that typing text would be torturous or impossible.

Lastly, keyboard command applications cannot disrupt or disable other products' activated features that are identified as accessibility features, where those features are developed and documented according to industry standards. Also, activated accessibility features cannot be disrupted or disabled when their application-programming interface has been documented by the manufacturer and is available to the product developer. This becomes an issue for applications development in selecting keyboard equivalents and for configuration management.

User Focus

In navigating for accessibility, it is important that a well defined on-screen icon be provided that moves among interactive interface elements as the input focus changes. The icon must be programmatically exposed so that assistive technology can track focus and focus changes. Indeed, standard screen metaphors that are normalized across operations provide easy navigation for the disabled and are helpful to everyone learning to navigate new software.

Be sure to make sufficient information about a user interface element, including the identity, operation, and state of the element, available to assistive technology. When images, icons, or bitmaps are used to identify controls, status indicators, or other software programmatic elements, the information conveyed by the image must also be available in text. Furthermore, the meaning assigned to those images needs to be consistent throughout an application's performance.

Textual information must be provided through operating system functions for displaying text. There is minimum information that must be made available: text content, text input caret location, and text attributes.

Color Usage

Colors selected for user interfaces can be an issue for people with limited sight (contrast) and color-blindness. Therefore color-coding should not be used as the only means of conveying information: indicating an action, prompting a response, or distinguishing a visual element. If color is used, then another color-neutral textual method should accompany it. For instance, instead of directing the user to click on the green button to complete the transaction, direct the user to click on the green "Done" button to complete the transaction.

Applications should not override user-selected contrast and color selections and other individual display attributes. This is important since adaptability tools for people with poor vision allow the user to select the high-contrast color that works best. When a product permits a user to adjust color and contrast settings, it provides a variety of color selections capable of producing a range of contrast levels.

Screen Motion

When animation is displayed, the same information should be displayable in at least one non-animated presentation mode for the user's option. To prevent a type of neurological seizure that can be initiated by the blinking on televisions and computer monitors, software should not use flashing or blinking text, objects, or other elements with a blink frequency greater than 2 Hz and lower than 55 Hz.

Filling Forms

All electronic forms must allow people using assistive technology to access the information, field elements, and functionality required to complete and submit the form, including all directions and cues. This means that screen readers and other devices should be able to read through the form and the text input, and that software and input screens can be navigated solely by keyboard commands. Visually impaired persons with blindness and low vision use screen readers. Purchasing a screen reader and testing software assures that it is accessible for the blind under Section 508 requirements. Of the screen readers available, two have emerged as clear market leaders, and both are used by federal employees: JAWS for Windows by Henter-Joyce, a division of Freedom Scientific, and Window-Eyes by GW Micro Inc.

Functional Performance Requirements

The regulatory standard also includes some functional performance requirements that software and system must address. At least one mode of operation and information retrieval that does not require user vision must be provided, or blind or visually impaired people must be given assistive technology. Providing keyboard equivalents and being screen-reader compatible for the text-oriented software meets this requirement. Remember that if diagrams, graphics or video clips are used text and/or audio equivalents must be available, too.

At least one mode of operation and information retrieval that does not require visual acuity greater than 20/70 shall be provided in audio and enlarged print output working together or independently, or support for assistive technology used by people who are visually impaired must be available.

For those that are deaf or hard of hearing, at least one mode of operation and information retrieval that does not require user hearing is to be provided, or support for assistive technology must be provided. It is important to remember this when including gongs or other sounds to alert for input errors or process errors. A visual equivalent should also be displayed. One similar frequent problem area in commercial software is that the blind user may know that a pop-up error messages has occurred, but the screen reader can not read the error message because the pop-up is an image and not text.

For the hard of hearing, where audio information is important for the use of a product, at least one mode of operation and information retrieval shall be provided in an enhanced auditory fashion, or support for assistive hearing devices shall be provided. And likewise, for those with problems speaking, at least one mode of operation and information retrieval that does not require user speech must be provided

Lastly for those dexterity problems, at least one mode of operation and information retrieval that does not require fine motor control or simultaneous actions and that is operable with limited reach and strength must be provided. People that have special adaptive appliances usually can plug those into a keyboard interface so the keyboard equivalents go a long way towards meeting this requirement on standard desktop PCs and terminals.

Other Parts of the Requirements

This article discusses the software development requirements of the Section 508 regulations. The regulations also has specific sections:

- Web-based information and operations.
- Telecommunications products.
- Video and multimedia products.
- Self-contained, closed products.
- Desk-top and portable computers.
- Information documentation and support.

The entire regulation and further information are available at www.section508.gov

The requirements of Section 508 are to be met in all systems (unless otherwise exempted), and to be available to all users, not as customizations for just those specific users with disabilities. The idea is that the information should become universally accessible. While this may appear to be add-on requirements now, I believe that it will become normal for all software to offer multi-modal access so that it can be used under many different conditions by anyone. Our challenge is to build these requirements into our software and systems development procedures and methods to begin getting the benefits or universal access.

Status of Implementation

The regulatory standards were issued in December 2000, so the 508 regulation takes effect by June 21, 2001. For purchased items, the Federal Acquisition Regulations are being modified. The industry and government agency officials in information technology, procurement, human resources, and technology are being briefed. Additional information on policies is also available at www.section508.gov ◆

About the Author

Keith Thurston is assistant to the deputy associate administrator in the Office of Information Technology, part of the Office of Government-Wide Policy at the U.S. General Service Administration. He works with a number of government-wide initiatives and groups to help formulate technology guidance, direction, and policy. He has worked with the Federal CIO Council since its inception in 1996 and focuses on the new policy issues as the technology evolves. Federal IT Accessibility Initiative implementing Section 508 Government-wide is the latest focus area for his office. Formerly Thurston was with U.S. Treasury and IRS working in technology and network communications for 14 years. He has a bachelor's in business and a master's of business administration in telecommunications.

1800 F Street NW, Room 2239
Washington, DC
Voice: 202-501-3175
Fax: 202-501-2482
E-mail: keith.thurston@gsa.gov

"I didn't want to pay to use somebody else's computer, so I decided to design my own."

Stephen Wozniak

"The internet is the world's largest library. It's just that all the books are on the floor."

John Allen Paulos

"People in charge are the last to know when things go wrong."

Alan J. Perlis

Streamlining Brings Oracle Big Savings, Better Service

Steven R. Perkins
Oracle U.S. Federal Government and U.S. Financial Services

The question Oracle asked itself was—once Oracle became an e-business, would it realize enough margin improvement to save a billion dollars? The answer turned out to be no. The company was going to save a lot more. This article covers some of the major internal changes Oracle made involving e-mail globalization, network consolidation, and running integrated internet-based applications. Oracle's experience translated into significant cost savings, but more importantly, better customer service and improved commercial products.

For the past two years Oracle has been changing its internal technology to streamline redundant communications and improve integration among core business systems. Following the desktop application revolution where spreadsheet data integrate easily into a word processing program or a slide presentation, Oracle has taken this same approach into its business applications. This required taking a hard look at its e-mail system, network structure, and use of Internet-based applications.

Initially, the company spent time scratching its head evaluating its current systems. In 43 data centers, Oracle had 70 different computer systems and 70 databases in 70 different countries in a client-server environment with some 40,000 desktops[1]. Its purchasing system could not identify the best suppliers by price, quality, and other metrics. The same was true for human resources and sales data.

On top of the lack of communication among similar and redundant business systems, a lack of integration among the core business systems themselves severely compounded the issue. These core business systems were a patchwork of programs from different vendors. This meant spending a tremendous amount of consulting dollars to make them talk, let alone the internal resources required to run the individual systems. Oracle's revelation was that at best it was not practical and at worst, it probably was not possible to make all these systems work together.

To change its technology in a way that would bring dramatic margin improvements, the company determined it would need to build a global system and defragment data. To keep up with the times, Oracle had to change how it ran its own business and it had to change its own products. The first tool Oracle incorporated to bring it all together was the Internet. This architecture provided a low-cost form of global communication that allowed the company to centralize network complexity and still distribute information worldwide. Oracle rewrote its databases and software-development tools to run in a three-tier Internet environment. The company rewrote its applications products too. Finally, Oracle implemented the technology internally.

After committing to an Internet environment, Oracle consolidated its Wide Area Network. Information technology (IT) personnel reduced the 43 worldwide data centers to two. One is located at its headquarters, in Redwood Shores, Calif. The backup is in Colorado Springs, Colo. Oracle has one global database for each major business function, such as sales and accounting.

Some of the tangible results from the centralization effort included 250 fewer IT staffers, 2,000 fewer servers, an 80 percent reduction in leased space for computer operations, and an overall estimated $200 million savings in IT costs for fiscal year 2000. Oracle is anticipating keeping total IT spending down to $300 million, which is half the amount spent in 1999. While the end results are more than worth the efforts, making it actually happen took both tough decision making and internal selling.

Global E-Mail Consolidation

While centralization, globalization, and consolidation plans were going on behind the scenes, Oracle needed to visibly demonstrate proof to employees and to resistant individual data owners that change was a good thing. Since e-mail performance would be highly visible, the company consolidated all of its e-mail systems. While this better and cheaper change was easy to sell, actually making it happen was challenging.

Oracle employees have many roles. Thousands of employees who use UNIX machines prefer certain e-mail clients. The Oracle sales force lives on laptops and prefers different e-mail clients. Some employees telecommute and access their e-mail using one client at work and another at home. The Oracle Data Center requires support for an industry standard e-mail interface that allows employees to use the client most appropriate to their jobs. Use of industry standard e-mail clients also avoids incurring additional costs to build and maintain custom clients to access messages.

Prior to the transformation, Oracle's worldwide e-mail "system" was an amalgam of 120 message stores, 50 data centers, and 97 mail servers supporting 33,000 user accounts on multiple platforms (See Figure 1, page 20) [2, 3]. Efficiency and reliability suffered from the large number of potential points of failure in the architecture as well as from recurring interoperability challenges. The company's existing messaging system actually consisted of many smaller systems. Not all servers ran the same versions of messaging software, not all servers were on the same platform, and the servers themselves were geographically scattered. There were 120 message stores worldwide. The system required 60 administrators because there were multiple systems all over the world. Furthermore, the Oracle system did not consist of a single domain (e.g. oracle.com), but sub-domains of oracle.com based on country (e.g. us.oracle.com). The result was an existing system that was inefficient to administer and costly to maintain.

Oracle's IT department decided to consolidate the various subsystems into one company-wide messaging system. While this reduced the amount of hardware, running a company-wide, enterprise messaging system on a minimum number of servers required software designed for that environment. IT researched available

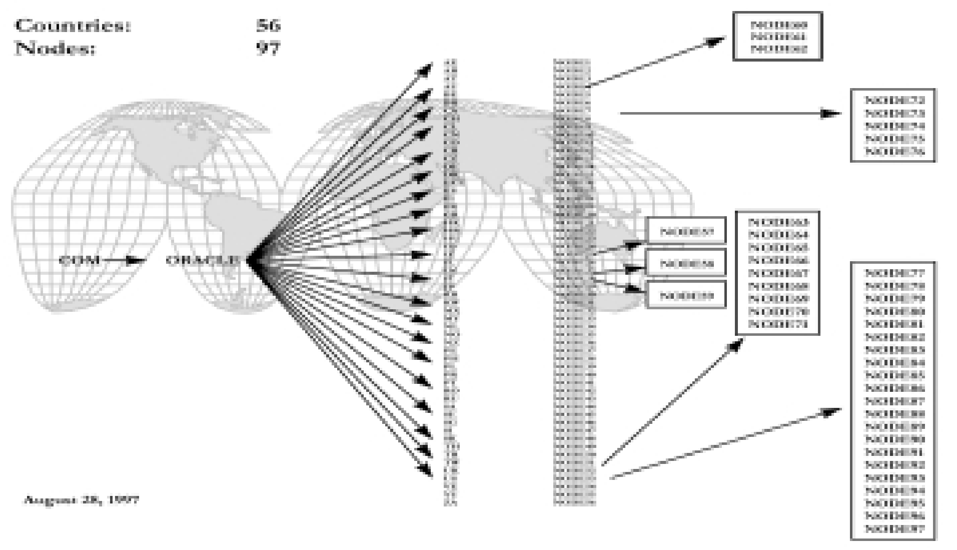


Figure 1: *Historical Oracle Messaging Domain Tree and Nodes*

software and determined to install an Oracle Email Server (OES) middle tier with Oracle8*i* databases as message stores, and have all employees use an Internet Message Access Protocol version 4 (IMAP4) Internet browser as an email client. The new system would allow the use of standard e-mail clients and enhance performance.

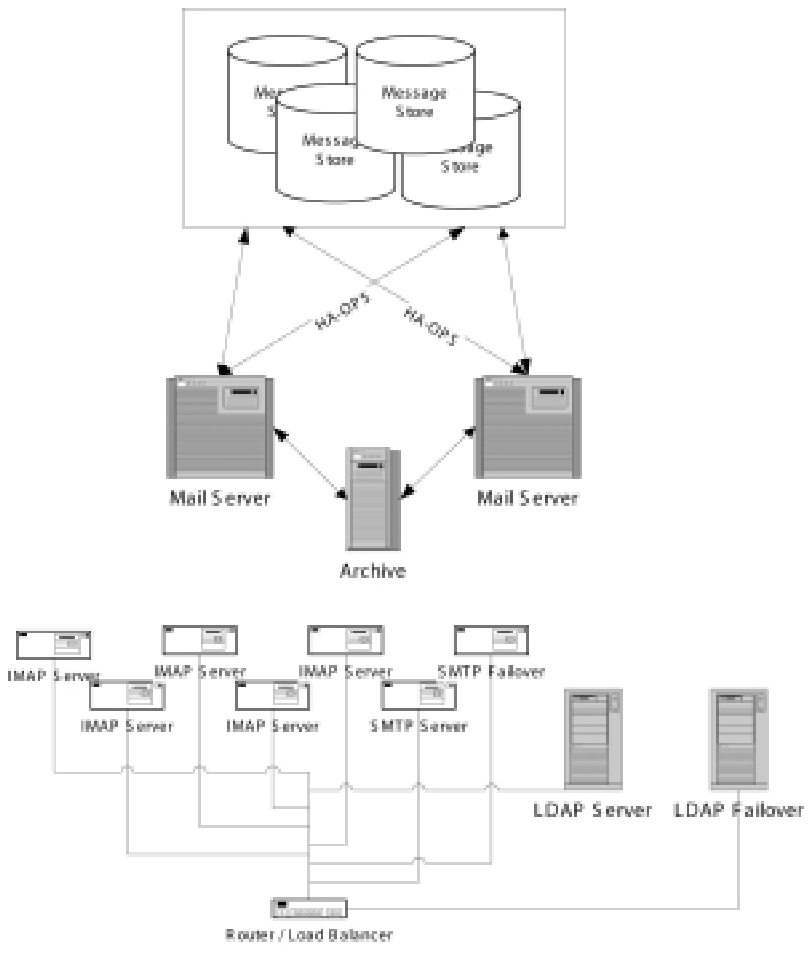
Among its desired qualities, OES is a highly scalable, open standards-based solution for very large corporate and Internet service provider customers. It allows users to easily access messages using any IMAP4- or Post Office Protocol version 3 (POP3)-compliant client. Its deployment cost per user is reduced because it increases the number of concurrent users that can be supported on the same machine. These qualities allowed the company to meet its goals: minimum hardware, simple deployment, and low cost of administration.

The money Oracle saved through less hardware investment could be used to purchase more powerful machines where needed in the system. The system is then able to support a larger user population because the servers are used around the clock. Administration demands also drop. The new messaging system only requires administration for four message stores on two machines at one physical location (See Figure 2). Administration is also only required for a single release of software on a single technology stack, easing the task considerably. System changes can be made more quickly. Administrators are able to simultaneously manage any of the system's component hosts, such as message stores and IMAP servers, from any point on the network.

The new system serves all the employees different needs. All messages can be controlled and administered in a central place to avoid synchronization problems. Employees who telecommute see the same inbox on their work systems as they do on their home systems. Furthermore, employees have the flexibility to change messaging clients when necessary.

Another benefit of Oracle's new consolidated messaging system is a global lightweight directory access protocol (LDAP) directory service. OES works with any directory that is compliant with the LDAP standard. This means that any LDAP-based client can access information in the directory, and directory information can be easily synchronized with any other LDAP-compliant directory. The LDAP product Oracle uses contains two nodes in the data center that each contains employee information. The directory data can be updated on either node. Employee searches are also easier. Users no longer need to look up employees by country, but just by name in a single worldwide directory. Misaddressed e-mails are corrected immediately; no message is bounced. Multiple languages are also supported, meaning employees in more than 96 countries around the world can use standards-based

Figure 2: *Architecture of New Oracle Single Instance Messaging System*



clients to access e-mail in their native languages.

The Oracle messaging system is a three-tiered architecture that can grow with a continually growing user population and mail storage needs (See Figure 3). To provide fault tolerance, the messaging servers containing the data would be high availability clusters with hot backups in a different geographic location. The messaging workload is partitioned into two distinct tiers. The middle tier acts as a protocol negotiator, allowing the translation of LDAP, IMAP4, POP3 and SMTP protocols into direct database queries. Each middle tier server multiplexes the Internet protocol requests over a relatively small number of database connections. This middle tier of protocol listeners work independently of each other so servers can be added one at a time as growth dictates, providing linear scalability. No data reside on this tier, so the demands for nearly zero down time can be relaxed as surviving servers fill in for a failed node.

Oracle's goal was to reduce the cost of maintaining the messaging system while meeting the messaging reliability, scalability, and performance needs of a company consisting of more than 43,000 employees. In calendar year 2000, the goal has been achieved; annualized e-mail system costs declined by $13 million, reliability is approaching "five 9s," worldwide operations are supported by a single corporate domain, and one-second message delivery times are standard.

Along with the dollar savings, Oracle also has a system that is easier to administer, support for standard clients to meet the needs of a mobile employee base, and a fault tolerant system that can grow with the needs of the company.

Many government agencies and corporate enterprises can achieve similar savings and performance improvements in their e-mail systems. Organizations may set targets well below the 50-to-1 e-mail server reduction that Oracle has already achieved, but huge potential savings would still accrue. Some organizations may be able to achieve high degrees of consolidation along the lines of the Oracle model.

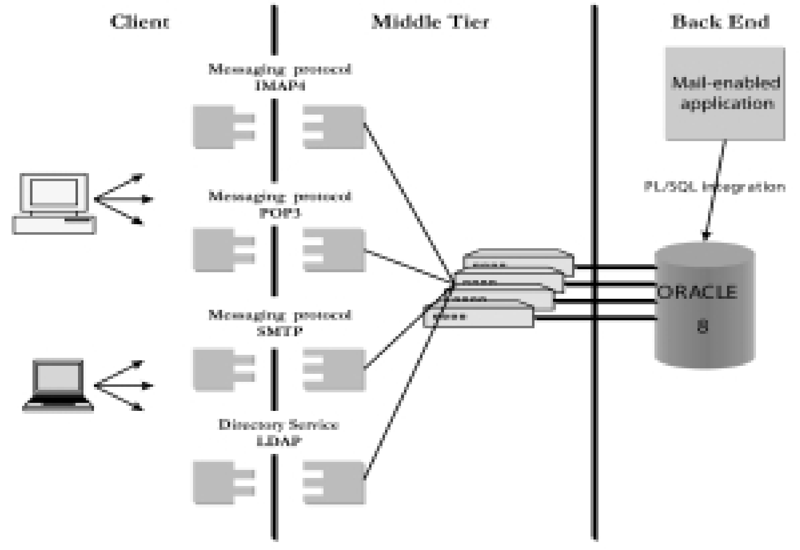
Consolidating the Network

For Oracle these changes — leveraging the Internet and globalizing e-mail — meant a heavy emphasis on redesigning its Wide Area Network. Previously, each of the Oracle subsidiaries throughout the world ran localized applications within each country's data center. Oracle's existing network at the time was not designed to meet the demands placed by running global applications from a single data center [4].

The biggest impact to the network was probably the most obvious: to have adequate capacity on all WAN links to carry the increased traffic load. The second biggest impact was one of availability. As users become increasingly dependent upon the WAN to run their business, the network design had to be as resilient as possible.

Since users were accustomed to interacting with a local applications server, enjoying LAN performance characteristics, Oracle had to look to reduce the WAN latency as much as possible. The ability to route traffic to a disaster recovery data center also became a network design

Figure 3 : *Messaging Workload Tiers*



Things to Consider in Consolidating an E-Mail System

Many companies offer hardware and software solutions for e-mail consolidation. Here are a few points to keep in mind while you do your research along with some of the results Oracle achieved in its solution:

— How many users can run on the system and still maintain rapid response times? An audited Oracle benchmark simulating 360,000 concurrent users on a single server with an average of over 13,700 messaging transactions per minute resulted in a response time of less than one second for a typical user sequence of sending or retrieving/reading email.

— What percent of server utilization reduction can be expected? Just by moving away from proprietary email clients, server utilization on the Oracle messaging system was reduced by an average of 15 percent to 20 percent, enhancing overall performance of the messaging system.

— What level of fault tolerance can be achieved? Oracle's messaging system node failure on the back end does not preclude data access because all data can be accessed from any node. Data integrity is maintained because committed work on a failed node is recovered automatically without administrator intervention and without data loss.

— In a node failure, how easily can load be balanced across surviving machines? Oracle's messaging system uses multiple servers, labeled the IMAP servers in the single instance architecture diagram, running OES in the middle tier, which are load-balanced. A failure of any of these machines simply means that users are redistributed among the surviving machines without loss of service. Similarly, Oracle Internet Directory is a highly available system.

— How will the system cut costs? Oracle achieved cost reductions by using software that can make the most of a smaller amount of powerful hardware.

consideration point.

The first task was to determine the actual traffic load that the single instance applications would place on the WAN links. E-mail, by its nature, is impossible to model using traditional modeling tools. The message sizes can vary greatly in size, from a 1KB message to a multi MB message with an attachment. The occurrence of these messages is also too random to model.

Oracle set out to actually measure the existing traffic load on the local mail servers. It found that traffic arrived at a fairly consistent number from .6 to .8 kbps. Not wanting to err on the low side and for planning purposes, the company used 1kbps per user, .8kbps of measured traffic + .2kbps for headroom. The per user number was multiplied by the number of users in each country by 1kbps to determine how much the WAN capacity needed to be increased for consolidating the email application. Therefore, a 200-person office would require capacity of 200kbps on the WAN links.

Oracle used 1kbps per user to determine where and how much the WAN capacity needed to be upgraded. Network resilience was increased, especially for its most critical locations, by carrier diversity and route diversity where possible, and by ensuring major backbone links do not traverse the same marine cable. The network had to be capable of automatically re-routing traffic to the disaster recovery data center located in Colorado Springs. To hold recurring WAN costs, a new network hub was created in Japan for North Asia locations. For example, Korea's mileage to the United States was reduced by more than 5,000 miles. There was a similar mileage saving for China as well. Oracle also converted two of the 64Kbps-satellite circuit at its India Development Center to a 768Kbps circuit over marine cables.

To mitigate earthquake concerns in Japan, Oracle split the network hub to two locations, Tokyo and Osaka, which are about 300 miles apart. The network hubs are located within carrier facilities, which are built to withstand earthquakes and other utility failures. With this design, the Japan-to-U.S. circuit's hub is out of Tokyo and the Japan-to-Singapore circuit's hub is out of Osaka. The North Asia countries normally have hubs out of Tokyo but can be switched to the Osaka facility in the event of a failure in Tokyo.

Also, in keeping with its design goals, the resilience between Singapore and Japan is built with high capacity links from different carrier's facilities. This design allows for Japan and Singapore to act as each other's backup and also maintain a full-time high-capacity link into the backup data center via the Singapore to Colorado Springs link. Similarly, the two sites in Australia back each other up and provide a direct link into the disaster recovery site, as well as a third backup route via Singapore being available.

The new Asia Pacific network met all of the design goals: increased capacity and resilience, reduced latency, and automatic routing to the disaster recovery site. In addition, Oracle was able to make all of these network changes without incurring too much of an increase in the run rate.

For example, Oracle previously had a 768kbps International Private Leased Circuit (IPLC) between Japan and the United States. Management upgraded to an Asynchronous Transfer Mode (ATM) service of 3mbps Sustained Cell Rate (SCR)/6mbps Peak Cell Rate (PCR) for approximately $6,000 per month less. That's quadruple the bandwidth for less money. Once the Japan hub was created, Oracle was able to provision ATM services to three out of the four North Asian sites, which were obtained at significantly less cost per kbps compared to the bandwidth provisioned over IPLCs [same as above] to Singapore.

Another example is the T-1 IPLCs into Australia from the United States. By changing to a carrier with a new presence in Australia, Oracle obtained the two T-1s for $9,000 per month less than the previous single T-1 service. The third T-1 to Colorado Springs is $9,000 per month less than the previous 768k circuit. This was done through renegotiating of the contract with the existing provider of the 768kbps service.

Within the United States, the network is based upon ATM and Frame Relay circuits. Oracle currently has 37 ATM sites and 44 Frame Relay sites for a total of 81 sites. Each site has at least two PVCs; one to headquarters and one to the disaster recovery site. Through new rates obtained by renegotiating with an existing U.S. network carrier and making use of a new network service offering, Oracle was able to double the bandwidth capacity to 70 of the sites and still realize a savings of $18,000 per month.

These changes did not happen over night. Oracle's methodical approach to using the Internet, creating a global e-mail system and re-designing its WAN to a more centralized model, laid the foundation, over which Oracle could then integrate its business systems and leverage ultimate cost savings.

Implementing Internet-Based Integrated Applications

The final implementation step was to roll out global Internet-based applications that allowed customers and employees to access information from a single database through an Internet browser, and to do transactions on their own such as on-line shopping or filing on-line employee expense reports. The cost of processing expense reports went from $60 down to $10, saving a total of $11 million in fiscal 2000. In total, the combination of other internal self-service applications cut employee costs by $150 million.

Through Internet customer self-service Oracle reduced the cost of supporting customers by $550 million in fiscal 2000. Several global databases allow customers to enter their own bug reports. This saves both time and money because individuals sitting on the phone do not have to serve as data entry go-betweens. It is estimated that a call handled by a Website is $20, while one handled by a person, which likely results in more follow up, costs $350. This change allowed Oracle to increase the chances of solving a problem by 100 percent and reduce costs by a factor of 17.

Oracle also saw huge savings in customer training. The company averaged a $250 cost per head to train customers at hotel seminars or conference centers. They moved training seminars online and dropped the cost per attendee to about $2. Oracle's margins on its education segment jumped to 41 percent in 2000 from 17 percent in 1999.

Oracle's infrastructure consolidation has certainly reduced operations and maintenance costs, with far fewer servers to administer. And, because web-enabled applications do not require installation of a software suite on individual clients beyond an e-mail-capable web browser, workgroup administration costs have declined dramatically. Some savings must also be apportioned to resourcing the supporting IT workforce and to establishing client help-desks where new or reengineered processes are introduced.

While Oracle is still in the midst of transformation and moving all its business processes to the Web, cost savings for new capabilities in knowledge management are still being defined. However, this area holds the promise for the greatest resource savings. Many cost reductions will derive from reengineering secondary processes that exist to enable legacy activity that is no longer required in a knowledge management environment (e.g., context-based content search routines that automatically return requested information, greatly reducing the requirement to maintain call centers).

For Oracle, what once was a pipe dream of consolidating stove pipe systems has now turned those client-server nightmares into fluid, Web-based information streams of seamless integration, making IT dreams come true while keeping bottom lines positive.◆

References

1. Karpain, Greg and Myers, Leeanne, Keep It Simple: How Oracle Consolidated Its Global Infrastructure into a Centralized E-Business Architecture, Oracle White Paper, September 2000.
2. Whitechurch, Charles, USAF Could Leverage Its Oracle Enterprise License To Enhance E-Mail Performance and Lower Total Cost of Ownership, Oracle White Paper, August 2000 .
3. Lee, Sandra, Cost Savings on Electronic Mail Through Consolidation, Oracle Paper #965.
4. Ellison, Larry, How We Saved A Billion Dollars, Oracle 2000 Annual Report.

About the Author

Steven R. Perkins has a long career history in the information technology industry, spanning sales and consulting for financial services and governments in the United States and Europe. He is also Senior Vice president of Oracle Service Industry's Consulting Practices. Before joining Oracle eight years ago, he worked for Arthur Young & Co., Booz-Allen & Hamilton, and the Department of Justice. Perkins manages a staff of 2,600 as senior vice president of Oracle Federal Government, U.S.Financial Services and Oracle Service Industry's Consulting Practices. He holds a master's degree in public administration from American University.

For questions call:
Tracy L. Strelser
Oracle Corporation
1910 Oracle Way
Reston, VA 20190
Voice: 703-364-5713
Fax: 703-364-3026
tracy.strelser@oracle.com
www.oracle.com

Coming Events

June 11-13
E-Business Quality Applications Conference
qaiusa.com/conferences/june2001/index.html

June 18-22
ACM/IEEE Design Automation Conference
www.dac.com

June 25-27
2001 American Control Conference
www.ece.cmu.edu/~acc2001

July 1-5
Eleventh Annual International Symposium of the International Council on Systems Engineering
incose.org/symp2001

July 7-13
2nd Int'l Symposium on Image and Signal Processing and Analysis ISPA'01
ispa.zesoi.fer.hr/

August 1-5
0HCI International 2001: 9th International Conference on Human-Computer Interaction.
1st International Conference on Universal Access in Human-Computer Interaction (UAHCI 2001)
hcii2001.engr.wisc.edu/

August 27-31
Fifth IEEE International Symposium on Requirements Engineering
www.re01.org/

September 10-14
Joint 8th European Software Engineering Conference (ESEC) and 9th ACM SIGSOFT International Symposium on the Foundations of Software Engineering (FSE-9)
www.esec.ocg.at/

Maintaining the Quality of Black-Box Testing

Patrick J. Schroeder
Illinois Institute of Technology

Dr. Bogdan Korel
Illinois Institute of Technology

Techniques to reduce cost and schedule can wreak havoc with product quality, especially in the software testing process. Shortsighted reductions can unwittingly result in the loss of long-term customers. In this article, the authors present an automated approach to reducing the cost of black-box testing while maintaining the quality of the testing process and software product. Experimental studies have shown a significant cost reduction using this automated approach.

Controlling cost, schedule, and quality in a software development project remains a challenging task. This type of control is difficult largely because of our inability to accurately measure attributes of the software development process, especially quality [1]. Measuring the quality of development processes and artifacts then relating them to final software product quality is not a process that is well understood. Under constant pressure to reduce cost and schedule, software engineers often use reduction techniques without fully understanding their impact on processes and final product quality.

Nowhere is this truer than in the software testing process. Software testing can improve software quality, but at a significant cost. It is not unusual for 40 percent to 80 percent of product development cost to be spent finding and fixing software errors [2]. Balancing testing cost and schedule with quality is difficult. Ad hoc reductions in the testing effort may bring short-term savings in cost and schedule. Unfortunately, these savings may be erased by quality problems discovered later in the product life cycle where fixes are expensive and a product's reputation may be badly damaged.

In this article, we present an approach that reduces black-box testing effort while maintaining the *quality* of the testing process. By quality, we refer to fault-detection capability of the test suite, or simply, how many faults a test suite uncovers. The type of software faults are those related to the correctness of program output; other types of software faults associated with other quality attributes such as performance, fault tolerance, usability, etc. are addressed in other testing process areas. Our goal is to create a *reduced* test suite that is smaller in size than the original test suite. The reduced test suite, however, is to maintain the fault detection capability of the larger, original test suite. This ensures that any savings in testing cost and schedule gained by executing the reduced test suite are not lost to expensive field repairs. To accomplish this, we perform additional analysis of the program under test. By using information gathered from the program specification and implementation, we can reduce the size of the original test suite in a controlled fashion that ensures that the original suite's fault detection capability is maintained.

While we believe our approach is applicable in a wide array of testing situations, in this article we focus on our initial efforts of applying our approach to black-box testing and the problem of combinatorial testing in data-driven programs. The following sections define the problem and current approaches; our approach, called Input-Output (IO) Analysis; details of implementing IO analysis; the results of experimental studies using our approach; and the conclusion.

Combinatorial Testing Techniques

In this article, we focus on black-box testing of data-driven programs. Black-box testing ensures that a program meets its specification from a behavioral or functional perspective and is typically performed without knowledge of software internals. Black-box testing can be applied to both control-driven and data-driven programs. In control-driven applications, outputs are determined by sequences of events or processing states. In data-driven applications, manipulation of data inputs and the relationships between data items determine outputs. Data driven applications, which typically have multiple inputs and outputs, include transaction-based systems, order-processing systems, application program interfaces, and many of the form-based applications common on the Web today.

Black-box testing techniques used for data-driven programs include equivalence-class partitioning and boundary value analysis [3]. Using these techniques, testers select test data values for each of the program's input variables. The tester must then consider how to test combinations of the selected test data values. It is important to test different input combinations, otherwise the result could be an unacceptable number of undetected software faults.

The most comprehensive approach to testing program-input combinations is referred to as combinatorial testing. In combinatorial testing, all possible combinations of the test data values selected for the program inputs are tested. (Combinatorial testing should not be confused with exhaustive input testing, which is testing every possible data value, valid and invalid, in every possible combination. This generates astronomically large test suites in all but trivial applications.) From a software quality perspective, combinatorial testing is desirable because it covers a large portion of the program's input space, resulting in fewer faults passed to the end-users of the product.

The challenge in combinatorial testing is managing the size of the test suite. In reality, combinatorial test suites grow very rapidly in size, frequently making the approach impractical. To fit within available resources and schedule, several approaches could be considered to reduce the size of the combinatorial test suite. Orthogonal arrays [4] and experimental design techniques [5, 6] have been suggested as ways of reducing the number of combinatorial tests. These techniques generate tests by combining test data for subsets of the input variables, e.g., tests may be generated for all pair-wise combinations of program inputs. These techniques do reduce the number of combinatorial tests dramatically, but their impact on the fault-detection capability of the test suite, especially when compared to combinatorial testing, is unknown. Random sampling [7] may also be used to reduce the test suite, but it leads to a reduction in fault-detection capability. In the next section, we present an automated approach that reduces the combinatorial testing effort while maintaining the fault-detection capability of the combinatorial test suite.

Combinatorial Testing Using Input-Output Analysis

Our approach performs additional analysis of the program under test. Information gained from this analysis is used to create a reduced test suite that is smaller than the combinatorial test suite (a subset of the combinatorial test suite). The reduced test suite maintains the fault detection capability of the larger, combinatorial test suite. We refer to this analysis as Input-Output (IO) analysis because it identifies the relationships between a program's inputs and outputs. Combinatorial testing can then focus on those input combinations that affect a program output, rather than considering all possible input combinations.

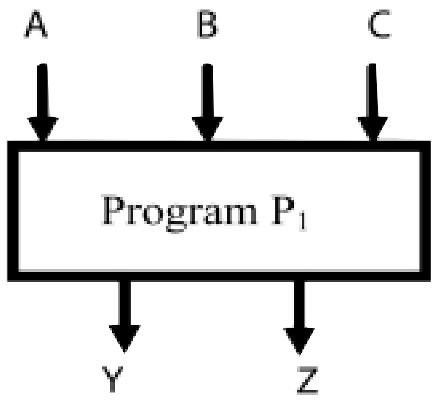


Figure 1: *Program P_1 -Multiple Inputs, Outputs*

For example, consider program P_1 in Figure 1. Program P_1 has three inputs (A, B, and C) and two outputs (Y and Z). To test this program using combinatorial testing, we would first select test data values for each of the program inputs using black-box test design techniques. Assume we select test data values for each of the three input variables, as in Table 1.

Table 1: *Selected Test Data Values for Program P_1*

Input Var.	Test Data Values
A	1, 2
B	North, South, East, West
C	TDC, BDM

Now consider how to test combinations of the input variables. For simplicity, assume there are no constraints among the input variables, and that we will execute combinatorial testing of the selected test data values. The combinatorial test suite generated from the test data values selected for the program inputs is listed in Table 2.

Our goal is to reduce the effort of combinatorial testing without reducing fault-detection capability. Our approach takes advantage of the observation that in data-driven programs not all inputs influence every program output. We find this to be a common occurrence in our study of this class of programs. We use IO analysis to identify relationships between program inputs and outputs, referred to as IO relationships. Based on the IO relationships combinatorial testing is reduced to testing only those input combinations that affect a program output.

For example, suppose we identify the IO relationships for program P_1 as in Figure 2. We find that output Y is a function of inputs A and C, and that output Z

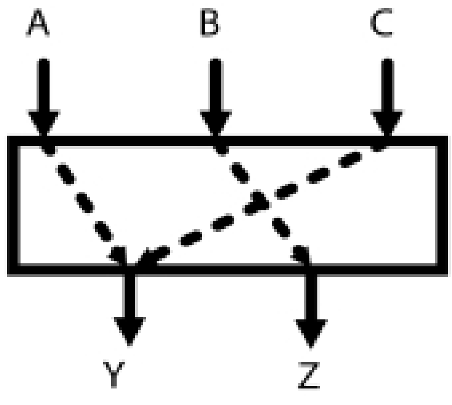


Figure 2: *Program P_1-Input/Output Relations*

is a function of input B, only. Since Y is a function of inputs A and C only, these two inputs will be used to generate combinatorial tests for output Y. Since there are four unique combinations of the test data values selected for inputs A and B, this would result in four tests. Similarly, output variable Z is a function of input B only, so tests for output Z will be generated from the test data values selected for input B, only. Since there are four test data values selected for input B, this results in four

Test ID	Input A	Input B	Input C
C_1	1	North	TDC
C_2	1	North	BDM
C_3	1	South	TDC
C_4	1	South	BDM
C_5	1	East	TDC
C_6	1	East	BDM
C_7	1	West	TDC
C_8	1	West	BDM
C_9	2	North	TDC
C_{10}	2	North	BDM
C_{11}	2	South	TDC
C_{12}	2	South	BDM
C_{13}	2	East	TDC
C_{14}	2	East	BDM
C_{15}	2	West	TDC
C_{16}	2	West	BDM

Table 2: *Combinational Test Suite Program P_1*

additional tests for a total of eight tests. However, notice that the test sets created from the perspective of outputs Y and Z can be merged into a test suite of size four, as in Table 3. The advantage of our approach is apparent; the size of the combinatorial test suite has been reduced from 16 to four tests.

By only generating tests for the input combinations that influence a program output, a reduced test suite is created. However, has the fault detection capability of the combinatorial test suite been maintained? For data-driven programs, it is easy to show that the fault detection capability is maintained if the IO relationships are correct. For example, if the IO relationships for output Y are correct, any software fault that causes an incorrect value at output Y is detected by entering some combination of the A and C test data values. There are only four unique combinations of the test data values selected for inputs A and C. In the 16-test combinatorial test suite, these four unique combinations of A and C are unnecessarily repeated four times: once for each test data value selected for input B. Clearly, from the perspective of output Y, there are repetitive tests in the combinatorial test suite. Repeating a test does not increase the fault detection capability of the test suite; it only increases its size. Similarly, removing repetitive tests from the test suite does not reduce its fault detection capability; it only reduces the size of the test suite. A similar analysis can be done for output Z.

Table 3: *Reduced Test Suite for Program P_1*

Test ID	Input A	Input B	Input C
C_1	1	North	TDC
C_4	1	South	BDM
C_{13}	2	East	TDC
C_{16}	2	West	BDM

Implementation of Input-Output Analysis

IO analysis is used to determine the relationship between a program's inputs and outputs. The IO relationships are used to reduce a combinatorial test suite by removing tests that are repetitive from the perspective of the program outputs. The fault detection capability of the test suite is maintained because repeating tests will not expose any additional software faults.

To use IO relationships to reduce the number of tests, we must ensure that the IO relationships are correct. If we reduce the number of combinatorial tests using IO relationships that are incorrect, we run the risk of removing tests that could expose a software fault. This would result in the reduced test suite having a lower fault detection capability than the combinatorial test suite. Therefore, it is important that we validate the IO relationships before using them to reduce the number of tests.

We validate IO relationships in much the same way that we validate any other software function. We identify an *expected* set of IO relationships from the program specification, and we compare it to the *actual* IO relationships as implemented in the software. This process is analogous to comparing a test's expected result with the actual result obtained from a program.

The first step in validating the IO relationships is to analyze the software's specification to identify the expected IO relationships. If one is using formal specifications, or rigorous component specifications, the IO relationships may be explicitly stated, or may be easily derived in an automated fashion. For other specification techniques, some additional analysis may be required to develop the expected IO relationships.

The overhead associated with identifying the expected IO relationships should be small, because the relationships are already being identified at several points in the software development process. For instance, software developers must identify IO relationships in the process of implementing the program. One cannot write code to produce a program output without knowing which program inputs are used to create that output. Similarly, software testers create expected results for each of their tests. Again, one cannot determine the expected result of a test without knowing which program inputs are used to create an output. The software development process could be modified to record the expected IO relationships as they are encountered.

The next step in validating the IO relationships is to identify the actual IO relationships implemented in the software. To identify these relationships, static analysis or execution-oriented analysis can be used. Both of these techniques can be automated.

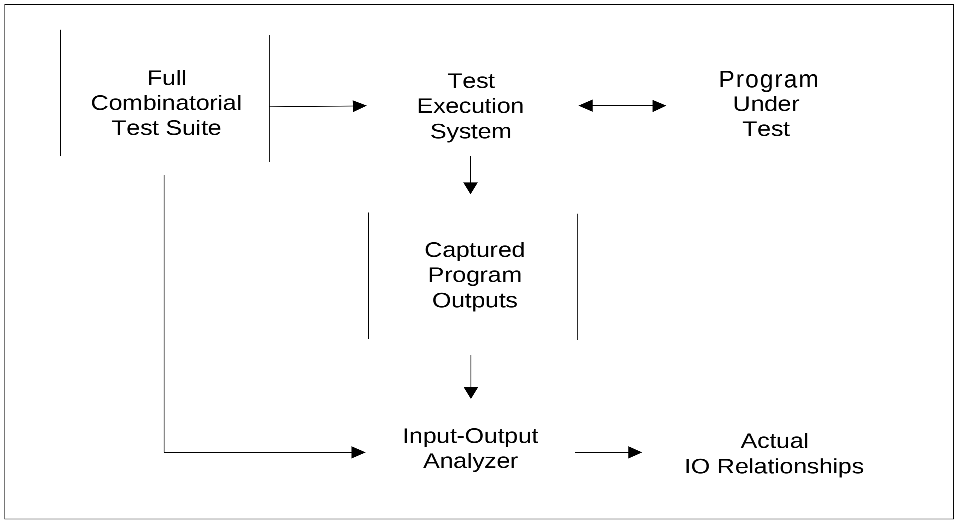
Static analysis is analysis of a program's source code. This analysis, accomplished using a source code specific tool, can produce information on a variety of the program's characteristics. One static technique used to determine relationships between inputs and outputs is referred to as Input-Output Relation Analysis [8]. This analysis uses a program dependence graph to determine which program inputs potentially influence a program output. Other program dependence analysis techniques used in code optimization, static slicing, and white-box testing, e.g. [9, 10, 11], may also be used to determine input-output relationships.

Execution-oriented analysis, as shown in Figure 3, is another technique that may be automated to identify actual IO relationships. This technique is based on program execution [12, 13] and does not require access to the program's source code. This technique does require, however, some type of test execution system, or test harness, to automated data entry and output capture. To determine relationships between inputs and outputs, the test harness executes the program under test many times altering only one input value. By observing changes in program outputs, it is possible to determine which outputs are affected by this input. This process can be repeated for all input variables in turn across a relatively large number of test data values. There is no guarantee that all IO relationships in a program will be detected using execution-oriented analysis. However, execution-oriented analysis can be used to identify all IO relationships exercised by a test suite such as the combinatorial test suite.

An advantage of execution-oriented analysis over static analysis is that execution-oriented analysis does not require the source code for the application. This makes execution-oriented analysis applicable in many testing situations, including testing of commercial off-the-shelf components. Execution-oriented analysis also has low startup costs, especially if test automation is already part of the test process. Another advantage to execution-oriented analysis is that the knowledge base and skill sets required to execute it closely match those of many software testers. Using static analysis to determine IO relationships, on the other hand, requires knowledge of the implementation's programming language and software internals.

The final step in validating the IO relationships is to compare the expected IO relationships identified in the program specifications to the actual IO relationships implemented in the software. This step is necessary if we are to maintain the fault detection capability of the reduced

Figure 3: *Execution-Oriented IO Analysis*



test suite. This is because the actual IO relationships could differ from the expected IO relationships. This difference could be due to different interpretations of the specification, to a fault in the software, or to complexities that arise during implementation that are not accounted for in the specification. To maintain the fault detection capability of the reduced test suite, these differences must be resolved before the IO relationships are used to reduce the combinatorial test suite

Experimental Studies

We have conducted several experimental studies using our approach. The goal of these was to understand the overhead involved with execution-oriented IO analysis and to determine the degree of reduction in testing effort.

We chose three different software applications to conduct the studies. The first application was the Total Return Report produced by a Windows-base personal financial software package. In testing a large system such as this, testing tasks are broken down by function area and assigned to testers. The Total Return Report represents a reasonably sized combinatorial testing task that may be assigned to an individual tester. The next application we studied was the Digital Trunk Configuration (DTC) software. This program provides an easy way to rapidly configure digital telephone trunks. The third application, the Liquidity Spreadsheet, was studied in conjunction with financial analysts who use spreadsheet programs extensively for financial modeling, reporting, and forecasting.

The procedure we used to conduct the experiments included the following steps. First, test data values for each of the application's input variables were selected using equivalence-class partitioning and boundary-value analysis. We then generated a combinatorial test suite from the test data values selected for each application using a test generation tool. The combinatorial test suite was executed automatically in a test execution system, and the outputs were captured. Finally, the inputs and outputs of each application were analyzed to determine the IO relationships. The reduced test suite was generated using these IO relationships. All studies were executed on a 250 megahertz Window-base PC. Table 4 lists the results of the experimental studies. Sizes are reported as the number of tests.

Our results show a drastic reduction in the number of tests required for these applications. This reduction does not come at the expense of a reduced testing quality. The automated techniques used keep the overhead of IO analysis low, making it a valuable combinatorial testing technique in many situations.

In implementing IO analysis, several costs must be considered. The development effort required to create the tools to perform IO analysis (test data generator, test harness, and IO analyzer) in our case, was minimal; although we do not want to underestimate the effort required to automate program execution is some situations [14, 15]. Creating the test harness had low overhead, but test harness execution times could get quite long for some applications. For programs that use a command line interface, such as the DTC software, IO analysis executes very quickly. For programs with a Graphical User Interface, as used in the Total Return Report study, data entry is complex and relatively slow. The speed of this process could be improved by using a faster computer, or by distributing the work across multiple machines.

This leads us to comment briefly, on how IO analysis may be used in the testing process. We assume a software production process that includes multiple pre-release versions of the software. Each version may incorporate fixes and incremental enhancements. For programs with reasonable IO analysis execution times, it is feasible to run the analysis with every pre-release version of the software. Executing IO analysis on every version of the software ensures that any changes in the IO relationships from one version to the next would be detected. In this scenario, the savings to the test team are largely in the effort to calculate and check the expected result for each test case.

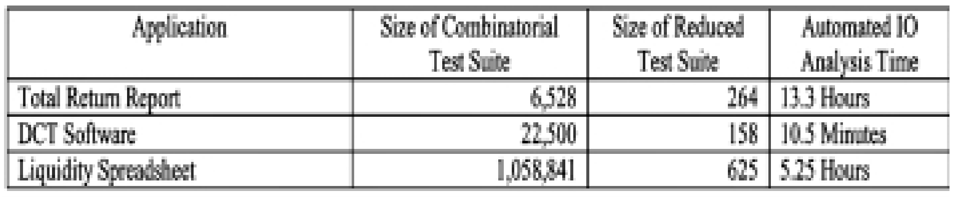
IO analysis executes a combinatorial test suite, but does not require determination of expected results, or comparison of expected results to actual program outputs. When IO analysis is used, the determination and comparison effort is only required of tests in the reduced test suite. In the case of the Liquidity Spreadsheet, the effort to determine and compare 1,058,841 expected results is reduced to the effort required for only 625 expected results. This represents a considerable saving to the test team considering that re-validation of expected results may be necessary with each pre-release version of the software.

For programs where data entry is complex and relatively slow, or where the combinatorial test suite is large, it will not be possible to execute IO analysis on every pre-release version of the software. In these situations, it is possible to execute IO analysis using a small subset of the combinatorial test suite. This subset can be constructed to validate the presence of the expected IO relationships in the software. The subset does not, however, ensure that other, possibly erroneous, IO relationships exist. IO analysis using this subset of tests could be executed on every pre-release version of the software. The reduced test suite created from this analysis will have a fault detection capability close to that of the combinatorial test suite. To ensure that the fault detection capability of combinatorial testing is maintained, IO analysis using a complete combinatorial test suite must be executed at some point. This would likely occur late in the product cycle when the software is stable and few additional software changes are expected.

Conclusion

We have presented an approach to combinatorial testing that reduces the testing effort while maintaining the fault detection capability of the combinatorial test suite. This approach can be used with little overhead for many applications especially where test automation is already in

Table 4 : *Results of Experimental Studies*



Application	Size of Combinatorial Test Suite	Size of Reduced Test Suite	Automated IO Analysis Time
Total Return Report	6,528	264	13.3 Hours
DCT Software	22,500	158	10.5 Minutes
Liquidity Spreadsheet	1,058,841	625	5.25 Hours

use. The experiment's results showed a drastic reduction in the testing effort of the data-driven programs. The low overhead associated with IO analysis and its ability to maintain the fault detection capability of the test suite makes it a valuable alternative to combinatorial testing.

The fault-detection capability of our approach is sensitive to correct identification of IO relationships. Automated IO analysis methods may not guarantee an identification of *essential* relationships in the presence of certain types of software faults. Therefore, it is important that software engineers validate automatically identified IO relationships in order to ensure their correctness. This extra effort may pay off later in reducing the testing effort while maintaining the quality of the reduced test suite. ◆

References

1. DeMarco, T., *Controlling Software Projects: Management, Measurement and Estimation,* Yourdon Press, Englewood Cliffs, N.J., 1982.
2. Kaner, C.; Falk, J.; and Nguyen, H. Q., *Testing Computer Software*, International Thomas Computer Press, New York, 1993.
3. Myers, G., *The Art of Software Testing*, John Wiley & Sons, New York, 1979.
4. Phadke, M. S., Planning Efficient Software Tests, CROSSTALK, Vol. 10, No. 10, pp. 11-15, October 1997.
5. Dalal, S.; Jain, A.; Karunanithi, N.; Leaton, J.; and Lott, C., Model-Based Testing of a Highly Programmable System, Proceedings of the International Symposium on Software Reliability Engineering, pp. 174-178, November 1998.
6. Burr, K., Combinatorial Test Techniques: Table-based Automation, Test Generation and Code Coverage, Proceedings of the International Conference on Software Testing, Analysis, and Review, October 1998.
7. Chen, T.Y. and Yu, Y.T., On the Expected Number of Failures Detected by Subdomain Testing and Random Testing, *IEEE Trans. Software Eng*., Vol. 22, No. 2, pp. 109-119, February 1996.
8. Korel, B., The Program Dependence Graph in Static Program Testing, *Inform. Processing Let*., Vol. 24, No. 2, pp. 103-108, January 1987.
9. Ferrante, J.; Ottenstein, K.; and Warren, J., The Program Dependence Graph and Its Use in Optimization, *ACM Transactions on Programming Languages and Systems*, Vol. 9, No. 5, pp. 319-349, 1987.
10. Horwitz, S.; Reps, T.; and Binkley, D., Interprocedural Slicing Using Dependence Graphs, *ACM Transaction on Programming Languages and Systems*, Vol. 12, No. 1, pp. 26-60, 1990.
11. Duesterwald, E.; Gupta, R.; and Soffa, M.L., Rigorous Data Flow Testing Through Output Influences, 2nd Irvine Software Symposium, Irvine, Calif., pp. 131-145, Mar. 1992.
12. Ferguson, R. and Korel, B., The Chaining Approach for Software Test Data Generation and Methodology, Vol. 5, No. 1, pp. 63-86, 1996.
13. Schroeder, P.J. and Korel, B., Black-Box Test Reduction Using Input-Output Analysis, Proceedings of the International Symposium on Software Testing and Analysis, pp. 21-24, August 2000.
14. Bach, J., Test Automation Snake Oil, *Windows Technical Journal*, Oct. 1996, pp. 40-44.
15. Fewster, M. and Graham, D., *Software Test Automation: Effective Use of Test Execution Tools*, Addison-Wesley, New York, 1999.

About the Authors

Bogdan Korel, Ph. D., is an associate professor of Computer Science at the Illinois Institute of Technology. His research interests include software engineering and automated software testing and analysis.

Illinois Institute of Technology
Department of Computer Science
10 W. 31st Street
Chicago, IL 60616
Phone: 312-567-5145
FAX: 312-567-5067
Email: korel@iit.edu

Patrick J. Schroeder is currently a Ph.D. candidate at the Illinois Institute of Technology involved in software testing research. Previously, he worked as a software developer and software tester on a variety of industrial projects, including seven years at AT&T Bell Laboratories. His research interests include software engineering, software reliability engineering, and automated software testing and analysis.

Illinois Institute of Technology
Department of Computer Science
10 W. 31st Street
Chicago, IL. 60616
Phone: 312-567-5150
Fax: 312-567-5067
E-mail: schrpat@iit.edu

"Computers are not meant to usurp human roles, but to aid an individual's work."

Donald D. Spencer

"Most industrialized nations today recognize that the computer is the cornerstone of their national defense and their national economics in the future."

C.W. Spangle

"Every educated person should have some understanding of the principles on which computers operate."

J. N. Snyder

"You know the definition of the perfectly designed machine ... The perfectly designed machine is one in which all its working parts wear out simultaneously. I am that machine."

Frederick A. Linderman

Managing the Invisible Aspects of High-Performance Teams

RADM Patrick Moneymaker
Ocean Systems Engineering Corp.

Dr. Lynn Carter
Software Engineering Institute

Many of the most important aspects of managing high-performance teams are counter-intuitive, yet we live in a culture that likes to reward its innovators and heroes even when clearly better solutions already exist. From new insights about metrics to the need to celebrate our successes, this paper describes a collection of five management lessons and provides useful references to sources for those needing implementation details.

If it were easy, everyone would be doing it. How many times have you heard this in the context of mastering a difficult task? The truth is, most challenges have a layer of complexity just beneath the surface that, once understood and applied, leads to the solution. The trick is to acquire a knack for seeing behind the everyday circumstances. The famous French aviation pioneer and philosopher, Antoine de Saint-Exuprey, had a key insight that applies: He contended that all that was most important in life was invisible. The invisible or counter-intuitive observations of management principals described here will hopefully provide a deeper understanding of difficulties previously experienced and some new ways to use traditional disciplines.

Thanks to a widely diverse career, I have a valuable perspective on two demanding yet diverse disciplines: military aviation and software development. As a Naval aviator I experienced the demands and thrills of flight training, carrier operations, combat, and ultimately duty as commander and flight leader of the U.S Naval Flight Demonstration Squadron, the Blue Angels. As a senior Navy officer and now as a senior executive of an engineering services firm, I have been responsible for software maintenance and development programs that ranged from the most strategic and complex to the most mundane.

During my aviation years I gradually developed into a seasoned pilot and tactician with each new level of skill supported by a solid foundation.

While involved in technology management I was thrust into a senior management role where I had little training and less experience.

Survival depended on applying my skills and instincts to the new setting. It was in this transitional setting – from maintaining situational awareness in air operations to maintaining effective software development program situational awareness – that I learned universal truths in management and team-building.

In each of the following insights, I will describe the software development context of my observation and my supporting aviation experience. Dr. Lynn Robert Carter, writing in Italics within the shaded boxes, will add references to others' works, including the Software Engineering Institute's (SEI) Capability Maturity Model® (CMM®). The two perspectives will hopefully add clarity to managing the "invisible."

Carter: *Behind each of Saint-Exuprey's invisible truths is a wealth of already captured knowledge. I'm fond of quoting Dr. R. W. Hamming in his 1968 Turing Award lecture where he states:*

> Indeed, one of my major complaints about the computer field is that whereas Newton could say, "If I have seen a little farther than others it is because I have stood on the shoulders of giants." I am forced to say, "Today we stand on each other's feet." Perhaps the central problem we face in all of computer science is how we are to get to the situation where we build on top of the work of others rather than redoing so much of it in a trivially different way. Science is supposed to be cumulative, not an endless duplication of the same kind of things.

I believe that all human efforts should be built on the lessons of those who came before us. To ignore the contribution of these giants shows our ignorance and tends to doom us to avoidable pain. Following each insight, I will provide supporting material and references to assist you in obtaining each insight's full benefit.

Determine Metrics Parameters First

Understand what you want to monitor *before* you ask for metrics. While this seems obvious, all senior managers face the same handicap: There is just not enough time in the day to understand the complexity of *every* project. Far too often senior management makes a general call for metrics (focused on the usual suspects, cost, and schedule) because they do not know any better.

The project manager who is concentrating on the day-to-day execution details is forced to drop everything to develop snapshot graphs showing all is well for the moment. This ill-conceived request causes more harm than good. It diverts the project staff's resources and more often than not gives senior management a false sense of security. A better approach would be an honest one-on-one conversation to determine the senior managements' information gaps so the executives can fully understand the project's critical path, and the project manager can understand the perspective from above.

Once this shared understanding is achieved, and here comes the invisible part, more meaningful "first derivative" measurements can be crafted. What do I mean by first derivative metrics? Simply those samplings that give warning of movement in time to make a correction. In aviation language, it means flying using the vertical speed indicator (VSI) *as well as* the altimeter. The VSI needle moves well before the altimeter, so if the object is to maintain a certain altitude, look for the trend on that instrument first and correct as needed. Likewise, while it is nice to know how much fuel (resource) is in the tank, a fuel flow indicator gives the added insight of burn rate. The fuel flow indicator is the first derivative instrument used in conjunction with the quantity indicator. Together they provide the view just

Capability Maturity Model® and CMM® are registered in the U.S. Patent and Trademark Office as a registered trademark of Carnegie Mellon University.

below the surface data.

In summary, when the call for metrics is sounded, first work toward a shared understanding of critical path, and then agree on what is measured and how. Anything short of these steps is a waste of time.

Carter: *In 1984 Victor R. Basili and David M. Weiss wrote a paper that introduced the Goal-Question-Measure Paradigm [1]. The heart of this approach is to start with the organizational goal that needs to be realized. From that goal, what are the questions that need to be answered in order to make corrections and steer to the goal? Given these questions, what measures need to be gathered in order to answer these questions, and how should this data be processed to obtain the needed situational awareness on which a meaningful decision can be made? Finally, given most people's overload, what mechanisms do we employ to ensure we take the time to examine the information and act when we should?*

The first challenge in applying this paradigm is the lack of clarity and alignment in an organization's goals. Without this, determining the right questions can be quite difficult. Even when clarity and alignment exist, the second challenge is agreeing to what questions should be asked and how numeric data, if it can be trusted, would support the decision-making process. A solution to all of these technical challenges leaves yet another challenge – the people. How do we change a heroic culture of option-based reactive decision making to one that values and honors the benefit of data and uses the data wisely? The benefits of these measures can only be seen when people both understand and use them. This requires knowledge, skill, and wisdom at all levels in the organization.

For more information, Software Measures and the Capability Maturity Model *[2], may be a useful resource. Those needing more sophisticated measures see* Measuring the Software Process: Statistical Process Control for Software Process Improvement *[3]. Anything less tends to degenerate into something appropriate for a Dilbert® cartoon.*

®Dilbert is a registered trademark of United Features Syndicate, Inc.

Follow a Disciplined Path

Discipline enables creativity and innovation. In one of my debates with Carter on the effects of the CMM on my programming staff, I offered that all code reuse, libraries, and such might inhibit the programmer's creativity. I did not want to have a crew of demotivated code benders who bemoaned the yoke of so much attention to process. Carter offered that quite the contrary was true. As I gave it more thought I began to see examples in military aviation that conveyed the message.

Considering the dynamics of carrier aviation it was easy to see that every step in the process of conducting a combat mission was, in itself, a small module of discipline. Briefing the flight, pre-flighting the aircraft and ordnance, launching from the flight deck, rendezvousing with wingmen, in-flight refueling en route to the target – all of these steps and many more had been practiced until they had become second nature. This training allowed the pilot to relegate much of the "house-keeping" chores to the subconscious, freeing up his immediate consciousness to meet the unexpected. Once in the target area and after the long chain of disciplined steps, quick thinking, creativity and innovation was always needed. Any number of unforeseen events could complicate the briefed plan. After adjusting for changes, discipline would again be needed as the pilot rolled into a steep dive to begin the bombing run.

The previous quote from Sir Isaac Newton is applicable here as well. Just as he had used the efforts of those who had blazed the trail before him, likewise, discipline and ever-improved process give an aspiring master a lofty platform from which to launch his creative assault on the unknown. So, process discipline properly applied, can be a catalyst for monumental discovery. In a world of ever increasing complexity, the need for the leverage of process discipline has never been stronger, nor the opportunity for breakthrough greater.

Carter: *It is easy to see how a senior leader and naval aviator could be concerned when first learning of the CMM and the additional actions it advocates. With no formal engineering background and no opportunity to learn how the benefits of using the CMM more than balance the costs, the admiral's reaction and comments were understandable. The challenge was to change his initial perception and begin the process of building an informed sponsor. At the heart of any improvement effort is a continuous effort of sponsorship development and sustainment [4].*

Nearly every leader has some aspect of his background where special training and discipline was required. Evoking those experiences and relating them to process discipline has proven very useful to me. The most difficult part is helping them see the benefits that come from separating the routine from the novel. When we learn to see routine problems and develop proven methods for addressing them, we obtain more predictable results and free people to invest their creativity on the novel aspects of their work. While it was true that combat pilots could employ common sense to figure out how to brief each other before a mission, pre-flight the aircraft, etc., what is gained and what is lost? Wouldn't it be better to save their creative energies for the novel aspects of the mission? The problem with common sense is its inability to tell us when it is about to fail us.

Senge's The Fifth Discipline *[6] provides numerous examples of the importance of capturing and leveraging lessons from past experiences. He uses the phrase "the learning organization" to describe a mode of operating that recognizes that common sense and the education people tend to receive are inadequate for the kinds of work we are called on to perform these days.*

At the heart of The Capability Maturing Model *[6] is the notion of a process asset library that contains the organization's standard software process, software life cycles, process tailoring guidelines and criteria, historical data for selection and estimation purposes, as well as documentation for familiarization, training, and use by professionals. Not mentioned in this book is what kinds of processes to capture, what processes to ignore, and the need to consider different representations of these processes and other materials for the different roles to be played and the different purposes for which it will be used. This is not really an omission, since the answer for one organization is likely to be very different from another.*

From my perspective, an organization

must find a balance between allowing the bright and insightful professionals the freedom they need to do their work and leveraging the insight of those that have come before them. This balance protects the organization from painfully relearning lessons. As I tell my SEI Defining Software Process students, if there isn't a horror story or a pot of gold story behind a process asset, why burden the organization with it? There is no need to formalize common sense when it works just fine. The challenge for every leader is to become familiar with the horror and pot of gold stories and ensure everyone sees how the organization's disciplines are really only there to help us avoid the former and leverage the latter.

Stand on Firm Ground

High performance teams need strong infrastructure. You have a big, important project, and you have brought in the best and brightest to form a team. Should you expect harmony with all of this excellence? Will they be one big, mature, happy family? It is unlikely.

Even with the ideal Myers-Briggs® personality match up, the most painstaking selection process, and a tailor-made project for your team, the need for a rigid but porous environment cannot be overstated. Rigid because you cannot stretch to a goal on a shaky ladder. Porous because one person's good idea or word of caution, even whispered, needs to be heard by the whole team.

The Blue Angels handle this need for a communication-rich structure by employing an extremely thorough briefing and debriefing process around "the table." Representatives from all departments in the squadron meet before every flight.

All administrative, logistical, and operational details are discussed. If a member has a question it is resolved on the spot. It is here that pilots mentally rehearse each maneuver, study the overhead photography of the airfield, and put on their game face. Because they take the time to do this *every* flight, they can then disperse at a show site with its hundreds of thousands of spectators and feel confident in the plan. During the performance, every aspect from manning the aircraft to deplaning after the show is videotaped. This videotape is the debriefing focus, which is conducted immediately after each show. These sessions are nothing short of exhaustive, but every detail is reviewed, and all comments are heard. So, it is a framework of structure and ritual that allows these top performers to be constantly "plugged in."

Within the software development domain, a similar construct is achievable. No, there does not have to be videotaped coding sessions, but the structure suggested by the CMM coupled with a little customized ritual to get people talking and in front of the issues can be very powerful.

Carter: *A painful mistake we often make is assuming that everyone is like us. One place where this assumption can be most awkward is in decision making. The book* Type Talk at Work *[7] provides a fascinating insight into 16 personality types, and how they influence success on the job. In particular, they describe a critical difference between how two different personality types – what the Myers-Briggs Type Indicator® calls "introverts" and "extraverts" – process information leading to a decision. Extraverts tend to process immediately and verbally, wanting to engage everyone in this process so a decision can be made. Introverts tend to process over time and quietly, wanting to think it through before speaking. Placing people with these two tendencies in the same room at the same time and asking them to make a decision often brings less than optimal results. Without care, a "decision" can be reached without the full benefit of all the team member's insights and experiences.*

Each Blue Angel demonstration begins and ends with time at the table. Before each demonstration, issues and concerns are raised and resolved. This is very similar to the Intergroup Coordination Key Process Area in the Capability Maturity Model *[6]. At the heart of these pre-project meetings are risk management activities. What concerns (risks) do we see? Which are likely to occur and warrant action? The better the group is at surfacing the risks, assessing their probability and impact, selecting which ones to address, and choosing mitigation or contingency actions, the better the project will be.*

After each flight, the Blue Angels reflect on what happened. From minor issues to surprises, the team tries to determine what should be done differently to improve things the next time. A number of authors have written about taking time to reflect on what we have done as an integral part of becoming better, yet few people actually take the time to reflect. In Habit 7, Sharpen the Saw, Steven Covey [8] relates a story of an exhausted wood cutter so intent on cutting down a tree that he cannot see the benefit of taking time to rejuvenate and sharpen his dull saw. Donald Schön [9] helps us see two kinds of reflection: reflection-in-action and reflection-on-action. The former addresses adjustment professionals make in real-time based on the dynamic and unpredictable realities of our world. The latter is done postmortem in order to set the stage for a better performance next time.

The Peer Review Key Process Area [6] strives to reveal issues as early in the life of the project as possible. Studies have shown that the longer a defect remains in a system under development, the greater the cost to remove it. Leveraging the differing perspectives of the team can uncover the genesis of defects well before it becomes obvious to the professional doing the work. In fact, the notion of continuous peer review is described in the method called "pair programming" [10], where the notion that the right two people working as a pair can often produce more than twice as much as either working alone.

Halt the Hero Cycle

Heroes are often the result of someone else's mistake. When a pilot plans a flight, he takes a host of things into account: distance to destination, fuel required (some cushion if the destination weather is bad), winds aloft, terrain elevation, and on and on. Why? Because anything short of a successful landing at the intended location is unacceptable.

What if the pilot were handed a flight plan that did not offer him a fuel reserve or worse yet, not enough fuel to get to the destination? Would he even take off on such a foolish plan? Thankfully the laws of physics are broadly understood and protect the pilot who would protest.

Myers-Briggs Type Indicator® is a registered trademark of Consulting Psychologists Press, Inc.

In the business world, a proposal for software development may not have had the due diligence required to get safely to the destination. Or margins were shaved to arrive at a desired cost. Yet time and again programming staffs assemble for project flights with no weather forecast and questionable fuel reserves. And their superiors expect them to land at their destination on time.

The invisible truth here is that heroes end up filling the gap in resources by over-extending themselves and their teams. If they survive, they are rewarded with a promotion, and the cycle is perpetuated.

There is another subtle point to be made about the plight of the hero. The project manager under pressure to deliver will rarely listen to a good idea if it is outside his implementation plan. He has literally no room to maneuver, since taking time to listen to something potentially helpful takes time he does not have.

So, how is the cycle broken to let remedy enter? The answer is not hard to see if you look outside of the project management environment where there is some discretion. It takes commitment, resources, and patience.

My firm has the pleasure of assisting the Software Engineering Process Office, a part of the Space and Naval Warfare Systems Center, San Diego, Calif., which recently earned CMM Level 3 certification. The staff, led by Beth Gramoy, has enjoyed such commitment from Dr. Kolb in her front office and has offered on-sight training, pilot programs, and generally carried this corporate goal into reality. It is this kind of organic resource and backing that is required to change an organization's culture.

Carter: *In his book,* Principle-Centered Leadership, *Stephen Covey [11] points out that there are numerous systems that fail to yield to pressure. These systems proceed based on natural laws or governing principles, whether we know about and appreciate them or not. Most pilots and their leaders know the maximum speed and range of their aircraft, so few pilots have to contend with unreasonable task assignments with regard to these basic aspects. A small minority of developers works for executives who know their teams' productivity capabilities per code type. The majority of managers act as if the teams' productivity is something that can be negotiated and influenced by applying appropriate motivation and pressure. Yet none would consider negotiating with the crew of a Boeing 747 in an effort to push its maximum speed past the sound barrier.*

Over and over, I hear leaders talk about achieving the SEI's CMM Level 3 as a critical business goal. When I ask them why they believe this is a critical goal, it becomes clear they do not really understand the principles on which the CMM was built or the natural laws it was designed to leverage. The main principle beneath the CMM is that continuous improvement and leveraging positive experiences help an organization avoid the black holes from which common sense is unable to protect it. Leaders striving to reach Level 3 without honoring this and other critical principles may obtain that level and may be allowed to compete, but neither the intent nor the real benefit of the model will be realized. The result is more work and paper with no real benefit, which damages the approach in the eyes of those who are forced to operate in such an environment.

Talking with the people who work with Gramoy is a wonderful experience. Here is a fine example of people who have taken the time to understand the hidden factors and discover the basic principles and natural laws. They are embarrassed by the praise, for the more they learn, the more they realize that they need to learn. Such professionals and leaders are a joy to find.

Enjoy the Success

Software engineers must learn to celebrate. Hidden behind every space shuttle launch is a grueling schedule that begins when the landing wheel stops rolling from the previous mission and ends when the launch button is depressed to start the next mission. In the intervening months between shuttle launches, every component and sub-component of the Space Transport System is checked and tracked. There are literally millions of details that come together at launch time. Then after all the painstaking attention to detail, all the diligence, all of the mind-numbing rehearsals, there is a celebration of the human spirit like no other on the planet.

While attending a shuttle launch I happened into a conversation with the bus driver who was taking us to the viewing site. All along the trip he would spout facts and trivia about NASA and the shuttle program. His enthusiasm was so contagious that the entire bus was lifted to a higher level of excitement and anticipation. I learned that he was a NASA engineer who had retired but still enjoyed the thrill of the launch. He embodied true passion and delayed gratification. He never once mentioned the tedious preparations for launch, the personal sacrifice, and long hours. However, he gushed with pride and the sense of accomplishment.

When I was flying with the Blue Angels, I was routinely struck by the audience's general lack of understanding of what it took to put on a demonstration. Some thought we improvised our maneuvers on the day of the show. Some thought the planes were somehow mechanically locked together in the diamond formation. Very few realized that every maneuver was the result of months of intense practice and that each maneuver fit together precisely to form the entire demonstration.

Often there would be an aircraft "gripe" that would require nationwide logistical coordination for the delivery of a part. The part would be delivered and installed, and the aircraft test flown before the next day's show. And yet, time after time, the show would go on. The maintenance crew proved to be magicians on a regular basis, pulling the rabbit out of the hat just before show time. Then the diamond formation would thunder into the air, eight engines in afterburner, rising majestically into the vertical and complete a loop on take off. Each member of the maintenance crew would follow the formation skyward and experience a penetrating satisfaction. All the pain was erased as each member of the team drank in the beauty of what they had helped achieve. It was a celebration, brief and pure.

Where there is no excitement at completion there is no passion. Conversely, where there is a cause worth celebrating, the passion is inexhaustible. But how can a project manager recreate a shuttle launch or a flight demonstration? The reality is (the invisible part) that within *all* human enterprise there are the seeds of satisfaction and accomplishment waiting to be nourished by a manager who is also a leader, one who can connect human endeavor with purpose. When a team can boast of delivering code on time and within cost, time after time, it will truly differentiate itself in the marketplace. That alone is worth a celebration.

Carter: *The beginning and the end are the most critical points. How these are handled are excellent predictors as to what will happen in between. One speaker I heard made an interesting observation that almost every good book on project management recommends including a project post-mortem to capture lessons learned. He goes on to ask why those same books do not advise starting each project by reading lessons learned.*

Acknowledging our humaness and employing proven methods to obtain our best is a critical job of every leader. The Blue Angels have a lot to teach us about how to build and leverage a high-performance team. It starts by finding precisely the right people to be on the team. This is accomplished by involving those from last year's team who will continue on into this year in the selection process. It continues with team-building rituals and practices. Not the meaningless waste of time we have come to abhor from ill-prepared facilitation "professionals," but activities specifically designed and used to build team respect and trust.

One of the best sources of materials available for building generic high-performance teams can be found in the Sibbet and Drexler Graphic Guide to Team Performance *[12]. While not every item in this guide will be appropriate for every team, it provides a wealth of tools and just enough context to help you understand why you might need to use one.*

From Newton, we see the reverence for those mathematicians who came before him and his belief that they helped him reach new heights. From Saint-Exuprey, we learn that the most important things in life are seldom easily seen. From the admiral, I have learned that there is more to be gained by leveraging the advances and accomplishments of others even when their domain appears to be unrelated. I believe there is much more the Blue Angels have to teach us all, if we will only take the time to see beyond the obvious to the invisible, and honor the lessons of those giants who have been this way before us.

Conclusion

At the end of the day, we will always find more goals than resources to support them, more data than understanding, more complexity than order. This is the nature of growth. The key is to keep one's bearing amidst continuous change, and always be respectful of the lessons of those who have gone before us. Hopefully the insights we have shared have pierced the veneer of visible reality and offered a view of important underlying issues that challenge each of us. Effective metrics needs a shared understanding, creativity needs discipline, and strong cultural foundations stabilize high-performance teaming. We need to focus on the systemic ills that surround our heroes and find the cause within our task worthy of excitement. ◆

References

1. Basili, Victor R., and Weiss, David M., A Methodology for Collecting Valid Software Engineering Data, IEEE Transactions on Software Engineering SE-10, Nov. 6, 1984, pp. 728-738.
2. Baumert, John H, and McWhinney, Mark S, *Software Measures and the Capability Maturity Model*, CMU/SEI 92-TR-25, Pittsburgh, Pa. Software Engineering Institute, Carnegie Mellon University, September 1992.
3. Florac, William, and Carleton, Anita *Measuring the Software Process: Statistical Process Control for Software Process Improvement*, (ISBN: 0-201-60444-2) Addison-Wesley Computer and Engineering Publishing Group, 1999.
4. Gremba, Jennifer, and Myers, Chuck, *The IDEAL^{SM} Model: A Practical Guide for Improvement*, 1997, www.sei.cmu.edu/ideal/ideal.bridge.html
5. Senge, Peter M., *The Fifth Discipline*,(ISBN: 0-385-26094-6) Bantam Doubleday Dell Publishing Group, 1990.
6. Paulk, Mark C., et. al., *The Capability Maturity Model*, (ISBN 0201-54664-7) Addison-Wesley Publishing Company, 1994.
7. Thuesen, Janet M. and Kroeger, Otto, *Type Talk at Work*, (ISBN 0-440-50699-9) Bantam Doubleday Dell Publishing Group, 1992.
8. Covey, Stephen R., *The 7 Habits of Highly Effective People*, (ISBN: 0-671-70863-5) A Fireside Book, Simon and Schuster, 1989.
9. Schön, Donald A., *The Reflective Practitioner*, (ISBN: 0-465-06878-2) Basic Books, Perseus Books Group, 1983.
10. Williams, Laurie A., and Kessler Robert R., All I Really Needed to Know about Pair Programming I Learned in Kindergarten, *CACM*, May 2000, pp. 108-114.
11. Covey, Stephen R., *Principle Centered Leadership*, (ISBN: 0-671-79280-6) A Fireside Book, Simon and Schuster, 1990.
12. Sibbet, David, and Drexler, Allan, *Graphic Guide to Team Performance* (ISBN: 1-879502-09-7) The Grove Consultants International, 1994.

About the Authors

Lynn Robert Carter, Ph.D., is a senior member of the technical staff at Carnegie Mellon University (CMU) in a dual appointment at the Software Engineering Institute and the School of Computer Science. The focus of his nearly 12 years at CMU has been on assisting technical workers, managers, and leaders in benefiting from adopting software technologies in the context of organizational overload. Previously, Carter worked for firms including Tektronix, Motorola, and a leveraged buy-out from GenRad.

Software Engineering Institute
Carnegie Mellon University
3857 East Equestrian Trail
Phoenix, AZ 85044-3008
Voice: 480-598-1247
E-mail: lrc@sei.cmu.edu

RADM Patrick Moneymaker USN (Ret.) is president of Ocean Systems Engineering Corporation overseeing full-spectrum engineering services that the firm provides for the Department of Defense, including software development, C4I systems integration, and program management. Moneymaker retired October 1998 from the Navy as rear admiral having held information technology positions that included J6, US Strategic Command and Commander, and Naval Space Command.

RADM Patrick Moneymaker USN (Ret.)
OSEC
3142 W. Vista Way
Oceanside, CA 92064
Voice: 760-722-3222
E-mail: pmoneym@osec.com

Dear CrossTalk

I have used back issues of CrossTalk often in my software process improvement work both at Xerox and Hughes. It is really helpful!

Delores J Harralson
Hughes Space & Communications

Dear CrossTalk

I would like to make available to CrossTalk readers a tool they can use at no cost. The successful software operation demands superior performance on the factory floor. Many organizations have made commitments to initiatives in SEI CMM, ISO 9000, or Six Sigma in order to deliver superior capability. Each of these initiatives has one thing in common, the practice of software inspections.

Experienced software practitioners and managers understand that software development is a process of experimentation involving the continuous discovery of technical information associated with the function, form, and fit of the software product. Peer reviews are an integral practice in the process of experimentation. Software inspections are the most rigorous form of peer reviews and an important element in the process of software product verification.

The National Software Quality Experiment (NSQE) collects core samples of software product quality from the Software Inspection Lab. The NSQE contains software inspection results collected during 1992-2000 from dozens of organizations that have supplied thousands of core samples to this national database. This database serves as a calibrating benchmark for those wishing to reason about software inspections metrics. A presentation on the NSQE was made at the STC Conference 2000 and a CrossTalk web-article on this appeared in December 1998.

The NSQE supplies the initial measurements and metrics needed by an organization to set its expectations. It contains the upper and lower control limits for minutes of preparation effort per defect, defects per thousand lines of code, lines inspected per conduct hour, defects per session, preparation/conduct effort, and return on investment.

People constantly ask about tools for software inspections. They also ask about the National Software Quality Experiment. This has led to the production of a simple, web-based mechanism to allow users to enter inspection measurements, view the derived metrics, and calibrate their results with those of the NSQE.

Participants are invited to conduct an NSQE assessment and to compare their software inspections results to the NSQE benchmark by visiting members.aol.com/ONeillDon/nsqe-assessment.html

Don O'Neill
Independent Consultant
ONeillDon@aol.com

Dear Mr. Alder:

I recently read your opening article in the January 2001 issue of CrossTalk, and something you said caught my attention. You mention some of the incredible advances that have been made in the field of modeling and simulation, and you contrasted these successes with some areas in which good simulations seem to be strangely absent. Those areas are the motivation of this correspondence.

Specifically, you wrote, "Then again, why stop with the simulation of physical objects? I can think of some human relationships that could use some practice. For those of you who have teenagers, I wonder if a simulation of parent-teen relationships would help. We spend a lot of time getting educated in technical subjects and leave to chance the most challenging aspects of our lives – human relationships."

A team of us at the Johns Hopkins Applied Physics Laboratory has been working on this very problem for the past four years, and we agree with you wholeheartedly. We realize that there is a large void in the practice that we engage in when it comes to human relationships, whether it is in the family or on the job. There are many "soft skills" rooted in human interaction that we don't train for in a virtual environment. Such as how to talk to kids about drugs, how to hire good employees, how to testify in court, how to effectively resolve conflict in schools, how to talk to angry customers, and the list goes on.

We have developed a very successful tool that effectivly trains FBI agents and allows them to practice detecting deception in potential suspects. It takes what they teach at the academy and incorporates it in a virtual environment through interaction with a simulated person named "Mike Simmen." This tool is now in use in over 70 federal law enforcement agencies across the country and in 20 foreign countries. It is making a significant impact in giving FBI agents practice in a human interaction environment, learning and practicing this very essential skill. In the end, it is saving lives and money because, like in the flight simulator, the mistakes are made in the simulation. The tool enables the agents to "find the right solution through trial and error and away from danger."

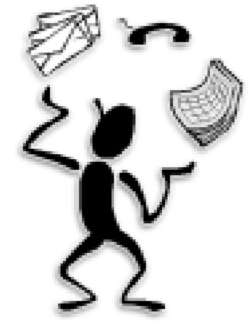
We are also about to begin developing other applications based on the same technology for other government sponsors.

Tim Frey
Johns Hopkins University
Applied Physics Lab
Simulated Human
Interaction Development

Dear

I'm having withdrawal ... not having CrossTalk to read since I left Puget Sound Naval Shipyard! The hard copy will be perfect to read on the Metro on my way to and from work!

Cathy Ricketts
NAVSEA



Software Odd to See

I received a message from Dave Cook, principal engineering consultant for the Software Technology Support Center, who was scheduled to compose the BackTalk commentary for this issue. He came down with laryngitis and has doctor's orders not to speak for two days. Now that would be an interesting sight. Asking Dave Cook not to chatter for two days is like asking Mick Jagger to line dance. He asked if I would fill in. Sure Dave, I know how it is when the voice goes; the mind can't be far behind. Have fun in New Orleans – at the Software Engineering Process Group Conference.

I called Pam, CrossTalk's managing editor, to ascertain the issue's theme. The reception on my cell phone was shoddier than a tube laden TV in a thunderstorm. I thought I heard her say the theme was "Software Odd to See, 2001." Certainly odd to me, but you know these imaginative journalists are always trying to be more creative than Ron Popeil at an infomercial convention. So I went with it.

What's odd in the software industry in 2001? I'll tell you what's odd. It's odd to see Y2K consultants, who scared the cash flow out of clients, flipping burgers at McDonalds because they invested their spoils in the latest dot-com craze. It's odd to see investors astonished by their high tech, no sweat, easy bet, Internet companies deflating faster than a helium balloon at a frat party.

It's odd to see engineers plaster their cubicles with Dilbert cartoons and grouse when they get passed over for management positions. It's odd to see project managers spend more time in process improvement meetings than with their customers. It's odd to see customers willing to pay more money for software from a company that has a higher capability maturity level. I thought they had better quality, leading to less defects, and thus reduced costs?

It's odd to see the Capability Maturity Model grow more appendages than a Siamese twin millipede. It's even odder to see experts try to integrate those independent appendages into a cohesive model. But most odd to see are customers clamoring for this new model like contestants during a Survivor immunity challenge.

It's odd to see software engineers more interested in the means (tools, language, process, etc.) rather than the results (customer satisfaction). It's odd to see software consultants who have never worked a complete project. Industry reference checks are about as scrupulous as a Clinton pardon review.

It's odd to see the most effective process in the industry – inspections – get the least amount of press. It's odd to see an industry, striving so hard to be an engineering discipline, naively jump on every technology bandwagon that drives through town. It's odd to see 75percent of the English language prefixed with the letter "e."

It's odd to see software processes constructed with all the demure expertise of Lucy and Ethel boxing chocolates. It's odd to see Software Engineering Process Groups, trying to improve organizational maturity, acting like Barney Fife on his fifth cup of coffee. Even odder is the rank and file blindly accepting KPA after KPA, like an inebriated Otis checking into jail, never questioning why or what for.

It's odd that after 10 years of rapid technological advances the words "paperless office" have left our vocabulary. After living through the Ada Dynasty, it's odd to see C++ being used on government projects. It's odd to see that a "CASE tool" is once again something you carry work in, to and from the office.

What's really odd to see is continued promotion of the best engineers into management. Didn't the Beatles teach us that writing great songs is not a good prerequisite for running the record company? It's odd to see software engineers promoted to management, suddenly and inexplicably bestowed with years of management wisdom. What's to learn? No need for training here.

Of course software is odd to see. It's intangible. You can see your keyboard, mouse, and monitor. If you splash water on your monitor you can see pixels. But you can't see software. Many attribute that fact to all that is odd in the industry.

If you really want to see something odd take these issues to your boss and ask for enlightenment. The rationalization will be as easy to understand as Jar Jar Binks, on Novocain, reciting Shakespeare. What an Odyssey.

– Gary Petersen, Shim Enterprise Inc.

A Free Workshop With A Beautiful Backdrop

Learn How to Get Software Requirements Right the First Time

The cost is FREE to U.S. government employees, and is located in the spectacular Wasatch Mountains of northern Utah.

Expert consultants from the Air Force s Software Technology Support Center, David Cook Ph.D. and Theron Leishman, will conduct **The Requirement for Good Requirements** workshop at **Hill Air Force Base** in Ogden, Utah, on **May 30-31, 2001**.

Here are some of the skills you will gain:
Understand the requirements lifecycle
Learn the attributes of good requirements
Understand the requirements analysis and elicitation relationship
Be acquainted with various requirements analysis techniques
Understand requirements validation and verification
Understand various types of requirements
Learn methods to elicit requirements
Understand documentation principles and issues

Plan to come early and enjoy what Utah has to offer: golfing, fly-fishing, hiking, biking, canoeing, rock climbing, river running, historic locations, and more. For travel ideas visit **www.utah.com.**

SPACE IS LIMITED. Act quickly. To reserve your place at the workshop, contact Debra Ascuena at 801-775-5778 or debra.ascuena@hill.af.mil. For additional information visit our web site at **www.stsc.hill.af.mil**.

Sponsored by the Computer Resources Support Improvement Program (CRSIP)

CrossTalk / TISE
5851 F Ave.
Bldg. 849, Rm B04
Hill AFB, UT 84056-5713

PRSRT STD
U.S. POSTAGE PAID
Kansas City, MO
Permit 34